Richtiges Sitzen am PC mit Hilfe eines intelligenten Bürosessels

Sabine Gross

DIPLOMARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2011

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 26. September 2011

Sabine Gross

Inhaltsverzeichnis

Erklärung iii

Vorwort vi

Kurzfassung vii

Abstract viii

1 Einleitung und Motivation 1

1.1 Dynamisches Sitzen 1
1.2 Sitzgelegenheit 2
1.3 Vorbeugende Software 3
1.4 Motivation 4
1.5 Übersicht 4

2 Verwandte Arbeiten 6

2.1 Interrupts 6
2.2 Veränderung des Sitzverhaltens 9

3 Konzept 11

3.1 Allgemeines 11
3.2 Sensoren 11
3.3 Intelligenter Bürosessel 12
3.4 Swopper 13
3.5 Kostenfunktion 15

4 Studie 16

4.1 Studiendesign 16
4.2 Rückmeldung 17
4.3 Training 18
4.4 Ergebnisse 20

5 Redesign **22**

5.1 Prototyp für Langzeitstudie ... 22

5.2 Verbesserte graphische Rückmeldung ... 22

5.3 Training ... 23

5.4 Anforderungen ... 24

6 Implementierung **25**

6.1 Allgemeines ... 25

6.2 Logik ... 25

6.2.1 Grundlegende Funktionsweise ... 25

6.2.2 Aufbau ... 26

6.2.3 Hauptkomponenten ... 27

6.3 Benutzeroberfläche ... 36

6.3.1 Grundlegende Funktionsweise ... 36

6.3.2 Allgemeines ... 37

6.3.3 Komponenten ... 37

6.3.4 Animationen ... 45

6.4 Datenspeicherung ... 46

7 Diskussion **48**

7.1 Intelligenter Bürosessel ... 48

7.2 Software ... 49

8 Ausblick **53**

8.1 Verbesserungen Prototyp ... 53

8.2 Aktives Büro ... 54

A Anhang ... 56

B Inhalt der CD-Rom ... 59

Literaturverzeichnis ... 62

Vorwort

There are 10 types of people in the world:
Those who understand Binary and those who don't.[1]

Während meines insgesamt 5-jährigen Studiums in Hagenberg habe ich viel Interessantes kennengelernt und meine Fähigkeiten und mein Wissen stetig erweitert. So habe ich mit *hungrigen Philosophen*[2] *gespeist*, mit sich *selbst erfüllenden Prophezeihungen*[3] die Prozessorleistung diverser Computer ausgereizt, in nach *unten wachsenden Bäumen*[4] nach „Früchten der Erkenntnis" gesucht und meine Backkünste durch die *Mandelbrotmenge*[5] verfeinert.

Nach dem Erlernen dieses Basiswissens war es mir vergönnt, neue Technologien kennenzulernen und weitere Erfahrungen in technischer als auch sozialer und kultureller Hinsicht sammeln zu dürfen.

An dieser Stelle möchte ich mich bei meinem Betreuer Prof. (FH) Priv.-Doz. DI Dr. Michael Haller bedanken, der mir bei diversen Projekten und dieser Arbeit mit Rat zur Seite stand. Weiters bedanke ich mich bei allen Mitarbeitern des *mil*[6], die ebenfalls immer hilfsbereit waren und konstruktives Feedback gaben. Sowie meinen *LipDub*[7]-Kollegen und Mitstudenten, welche dazu beigetragen haben, das Studium unvergesslich werden zu lassen (*Jacuzzi!*). Meinen Arbeitskollegen bei *Amatic Industries*, die mich in die Arbeitswelt eingeführt haben.

Der größte Dank gilt meinen Eltern, die mich immer unterstützt, an mich geglaubt und mir die notwendigen Freiheiten gegeben haben, damit ich zu jenem Menschen werden konnte der ich heute bin.

Alle geschlechtsspezifischen Nomen oder substantivierte Verben in dieser Arbeit gelten auch in ihrer weiblichen Form.

---

[1] 10 (eins, null) ist die binäre Äquivalenz zur dezimalen Zwei
[2] *Dining Philosophers Problem,* bekannte Problematik der Informatik
[3] Rekursive Algorithmen.
[4] Binäre Suchbäume bzw. Suchalgorithmen.
[5] Die fraktale Geometrie der Natur, Benoît Mandelbrot, Birkhäuser Verlag (Januar 1991)
[6] Media interaction lab, http://mi-lab.org/
[7] Eine Art von Musikvideo, http://vimeo.com/26154936/

Kurzfassung

Die vorliegende Arbeit beschreibt ein System, das auf Basis von Echtzeitdaten, die Sitzhaltung eines Benutzers erfassen kann. Das System dient dazu den Benutzer auf sein Sitzverhalten aufmerksam zu machen, um gesundheitliche Beeinträchtigungen durch schlechtes Sitzen zu vermeiden.

Diese Arbeit gibt einen Überblick über richtiges Sitzen und welche Produkte hierfür auf dem Markt existieren. Angefangen von Sitzgelegenheiten, die zur Verbesserung der Haltung beitragen über Software, welche zur Verminderung von gesundheitlichen Folgeschäden durch das Arbeiten am Computer eingesetzt werden kann.

Außerdem beschäftigt sich diese Arbeit mit der Informationsweitergabe (Interrupts) von Computerprogrammen, welche den Arbeitsfluss so wenig wie möglich stören. Anhand des entwickelten Systems werden zwei verschiedene Möglichkeiten aufgezeigt, wie die Information über das Sitzverhalten an den Benutzer weitergegeben wird.

Weiters wird beschrieben, wie ein handelsüblicher Bürosessel aufgerüstet werden kann, um die notwendigen Messungen über die Sitzposition vornehmen zu können.

Abstract

The present work describes a system which is able to track the sitting posture of a user according to real time data. This system serves to draw the user's attention to his/her sitting behavior so that the user can avoid health impairment caused by a false sitting posture.

This thesis covers the basics of good sitting behavior and supporting products which are available on the market. Starting with seating-accommodations which improve the sitting attitude to software which reduces consequential damage of wrong usage of the computer.

In addition to that this thesis looks into interrupting modality of computer programs which is less disruptive. The developed system in this work will present two different approaches to display the users sitting behavior.

Furthermore it is shown how a standard office chair can be upgraded in order to conduct the necessary measurement of the real time data of the sitting posture.

Kapitel 1

Einleitung und Motivation

Im heutigen Büroalltag verbringt ein Angestellter im Durchschnitt 50.000 Stunden sitzend während seines Arbeitsleben [26]. Ertel et al. [18] zeigen, dass durch die geringe Bewegung am Arbeitsplatz rund 40% der Büroangestellten an Rückenproblemen leiden. Außerdem kann es durch die statische Belastung von Rücken-, Schulter- und Nackenmuskeln zu Beschwerden der beanspruchten Muskeln kommen [8, 21]. Des Weiteren können Bandscheibenprobleme und Wirbelsäulenversteifungen auftreten.

Heutzutage ist der Einsatz von Computern in der Arbeitswelt nicht wegzudenken. Im Zusammenhang mit der Nutzung des Computers tritt ein weiteres gesundheitliches Problem in Erscheinung. Das *Repetitive Strain Injury Syndrome (RSI)* wird durch wiederholte und einseitige Belastung ausgelöst. Durch fortwährende Benutzung von Maus und Tastatur [9, 25] kann *RSI* verursacht werden. Im schlimmsten Fall erreicht *RSI* ein chronisches Stadium, welches nur mehr teilweise heilbar ist.

1.1 Dynamisches Sitzen

Schon als Kind wird man zu einem stillen und aufrechten Sitzverhalten erzogen, welches sich durch das ganze Leben fortsetzt, da es sich so in der Gesellschaft manifestiert hat. Neueste Erkenntnisse zeigen, dass diese statisch passive Sitzhaltung ein falscher Ansatz für gesundes Sitzen ist und zu Verspannungen und im schlimmsten Fall zu chronischen Beschwerden führen kann [10]. Um Muskel- und Wirbelsäulenproblemen vorzubeugen ist ein dynamisches Sitzverhalten anzustreben. Dynamisches oder aktives Sitzen scheint einen Widerspruch darzustellen, da man sich während des Sitzens nicht großräumig bewegen kann. Unter aktivem Sitzen versteht man allerdings, das häufige Wechseln der Sitzposition.

Neue Erkenntnisse zeigen, dass der menschliche Körper evolutionsbedingt nicht für langes Sitzen gebaut ist [17].

> *If motion is the price of muscles, pain is the price of lack of motion and muscle.*

Mit diesem Satz erklärt Egoscue et al. [17], dass nicht der PC der Grund für Schmerzen am Arbeitsplatz ist, sondern eine falsche Benutzung/Handhabung.

Ähnlich ist es auch in [10] beschrieben. So wird die Rückenlehne kritisiert, welche ein vernünftiges Arbeiten am Schreibtisch erschwert und gleichzeitig die stützende Funktionen der Wirbelsäule bzw. des Rückens übernimmt, wodurch die Wirbelsäule ihrer eigentlichen Aufgabe (Beckenkamm- und Rückenstütze) beraubt wird.

> *Bewegung ist Leben – Leben ist Bewegung. Ein Plädoyer für ein „lebendiges“ Sitzen.*

Neben dem „lebendigen“ Sitzen wird auch folgende Verteilung des Büroalltages vorgeschlagen:

- 60% Sitzen (lebendiges Sitzen),
- 30% Stehen und
- 10% Bewegung im Raum.

In [23] wird aufgezeigt, dass eine bewegliche Sitzfläche dem körpereigenen Bedürfnis gerecht wird und ein dynamisches Sitzen zur Folge hat. Dieses dynamische Sitzen hat neben den Bewegungen der Rücken- und Wirbelsäulenmuskulatur auch den Vorteil, dass die Blutzirkulation angeregt wird. Durch die bessere Blutzirkulation erfolgt eine bessere Sauerstoffversorgung, die einen positiven Einfluss auf die Aufmerksamkeit und Konzentration hat.

1.2 Sitzgelegenheit

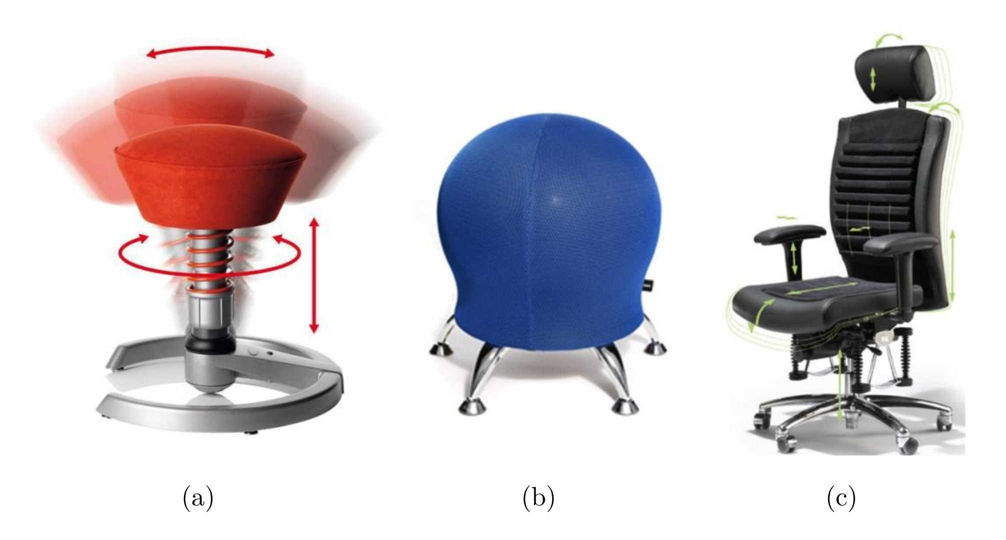
Neben den Erkenntnissen über besseres, dynamisches Sitzen gibt es Sitzmöbel, welche diese auch umsetzen. So haben verschiedene Hersteller unterschiedliche Systeme entwickelt, die für eine bessere Sitzhaltung sorgen sollen.

Abbildung 1.1 (a) zeigt den *Swopper*, der eine leichte Gegenbewegung zur Bewegung des Sitzenden erzeugt und somit verschiedene Muskelgruppen anregt [8].

In Abbildung 1.1 (b) ist der *Sitness 5* abgebildet. Unter dem Stoffbezug findet sich ein handelsüblicher, aufblasbarer Gymnastik-Ball. Dies ermöglicht ebenfalls ein bewegtes und dynamisches Sitzen

Die Firma *Haider* bietet verschiedene Modelle für ein dynamisches und

[8] Quelle: http://www.swopper.de



(a) (b) (c)

Abbildung 1.1: Das Bild (a) zeigt den *Swopper*. Ein Produkt von *Aeris*[9]. Bild (b) zeigt den *Sitness 5* von *Topstar*[10]. Bild (c) zeigt den *Bioswing 560 detenor* von *Haider*[11].

aktives Sitzen an, Abbildung 1.1 (c). Ein rhythmisches Pendeln aus und über die Körpermitte wird durch ein dreidimensionales Sitzsystem erzielt, wodurch ein gesundheitsförderndes Sitzen möglich ist [12].

1.3 Vorbeugende Software

Auch ohne Sitzgelegenheiten, welche einem bei einem dynamischen Sitzen unterstützen, kann man mit Software vor allem *RSI* vorbeugen.

Die bekanntesten Vertreter für *RSI*-Vorbeugende Software (käuflich zu erwerben) sind *Workpace*[13] und *RSI-Shield*[14]. Diese erinnern den PC-Benutzer in regelmäßigen Abständen daran, Pausen einzulegen. In diesen Pausen werden kurze Übungen angezeigt, welche ein Ausgleich für die andauernde Belastung darstellt. Neben den kommerziellen Softwareprodukten gibt es auch *Freeware* Produkte wie *Workrave*[15], *RSIBreak*[16], *Breaker*[17] und

---

[9] Quelle: http://www.swopper.de
[10] Quelle: http://topstar.de
[11] Quelle: http://www.bioswing.de/de/products/sitzsysteme/b560-detensor/
[12] Quelle: http://www.bioswing.de/de/bioswing/bioswing-sitzsysteme/rhythmus/
[13] http://www.workpace.com, Plattform: Windows.
[14] http://www.eagerplanet.com/rsishield, Plattform: Windows.
[15] http://www.workrave.org/, Plattform: Windows, Linux.
[16] http://www.rsibreak.org/, Plattform: Linux.
[17] http://davidevitelaru.com/software/breaker/, Plattform: Windows.

weitere. Wie auch die kommerziellen Produkte, erinnern auch die kostenlosen in regelmäßigen Abständen, dass der Benutzer eine Pause einlegen soll. Anders als bei den kommerziellen Produkten werden nicht bei allen frei erhältlichen Übungen vorgeschlagen.

1.4 Motivation

Es soll ein System entwickelt werden, dass die Vorteile der aktiven Sitzgelegenheiten und der vorbeugenden Software vereint. Es soll dazu dienen, den Benutzer über seine aktuelle Sitzhaltung zu informieren. Des Weiteren soll es den Benutzer, bei einer schlechten Sitzhaltung über einen längeren Zeitraum hinweg, darauf aufmerksam machen und geeignete Maßnahmen vorschlagen. Aufgrund der häufigen Computerarbeiten soll dies eine Software übernehmen, aber gleichzeitig soll es den Büroalltag nicht beeinflussen oder stören.

1.5 Übersicht

Kapitel 2 Verwandte Arbeiten gibt einen Überblick über die grundlegenden Themen die für diese Arbeit herangezogen wurden. Neben der Behandlung von Interrupts werden auch Arbeiten über die Veränderung des Sitzverhaltens mit Hilfe von Ausgabegeräten erörtert.

Kapitel 3 Konzept beschreibt die Mechanik, welche zur Messung der Sitzposition verwendet wird. Weiters wird die Berechnung erläutert anhand derer man das Sitzverhalten über einen Zeitraum bestimmen kann.

Kapitel 4 Studie schildert die Vorgehensweise um das richtige Rückmeldesystem für den Prototypen zu finden. Dafür wurde vorab eine Studie durchgeführt, welche drei verschiedene Systeme gegenüberstellt.

Kapitel 5 Redesign erörtert die Änderungen am Konzept, welche auf Basis der Studie aus Kapitel 4 getroffen wurden. Darüber hinaus werden Anforderungen festgelegt, die der zu entwickelnde Prototyp erfüllen muss, um für die geplante Langzeitstudie verwendet werden zu können.

Kapitel 6 Implementierung erläutert die Umsetzung der getroffenen Konzept- und Redesignentscheidungen und beschreibt den erarbeiteten Prototypen und dessen Aufbau. Dabei wird auf die notwendige Verwendung von gängigen Programmierkonzepten eingegangen.

Kapitel 7 Diskussion zeigt Lösungsansätze für Probleme die während der

Entwicklung des Prototypen auftraten.

Kapitel 8 Ausblick präsentiert mögliche Verbesserungen und Erkenntnisse, welche während der Testphase für die Langzeitstudie ermittelt wurden und für einen möglichen Produktprototypen in Frage kommen.

Kapitel 2

Verwandte Arbeiten

In diesem Kapitel werden grundlegende Themen erläutert, die Basis für den Design-Prozess waren. Für die geplante Software stehen vor allem Arbeiten im Vordergrund, welche sich mit der Art und Weise der Informationsaufbereitung für den Benutzer beschäftigen. Weiters werden Arbeiten vorgestellt, welche einen Ansatz bieten, wie der Benutzer zur Änderung seines Sitzverhaltens bewegt werden kann.

2.1 Interrupts

Als *Interrupt* wird jedwede Unterbrechung der derzeitig bearbeitenden Aufgabe angesehen. Beispiele für Unterbrechungen sind Telefon klingeln, Fragen eines Kollegen und elektronische Unterbrechungen wie E-Mail Benachrichtigungen. *Interrupts* stören den Arbeitsfluss und beeinflussen diesen nachhaltig. Während man die äußeren, nicht elektronischen Unterbrechungen kaum beeinflussen kann, gibt es viele Studien, wie man den Störfaktor der elektronischen *Interrupts* minimieren kann [1, 3, 6, 13].

Ein elektronischer *Interrupt* entsteht, wenn eine Software oder das Betriebssystem Informationen an den Benutzer weitergibt. Die Art der Unterbrechung bestimmt einerseits wie groß die Störung ist, andererseits wie hoch die Aufnahme dieser Information ist. Tabelle 2.1 zeigt, dass ein Optimum erreicht wird, wenn der *Interrupt* wenig störend, die Informationswahrnehmung hoch ist [6]. Diese Tabelle bezieht sich auf Benutzeroberflächen (*UIs*) im Webbereich. Dennoch kann diese Gegenüberstellung auch auf Desktop-Anwendungen angewendet werden, da auch hier Methoden wie das Dialog-Fenster und Statusleisten-Meldungen verwendet werden.

Des Weiteren ist auch relevant, in welcher Phase sich der Benutzer befindet, wenn er unterbrochen wird. Wie in [14, 15] beschrieben, wird eine Aufgabe in drei Phasen unterteilt: Planungs-, Ausführungs- und Evaluierungs-

phase. In der Planungsphase, wird die Aufgabe analysiert und ein Lösungsansatz erarbeitet. Bei einer Suchaufgabe im Internet, wäre die Planungsphase, das Erarbeiten der richtigen Schlagworte für die Suche. Die Ausführungsphase ist die Bearbeitung der Aufgabe anhand des erarbeiteten Lösungsansatzes. Auf das Beispiel der Suche bezogen, wäre dies die Eingabe der Schlagworte und das Aufrufen der Suchergebnisse. Die Evaluierungsphase ist die Auswertung der erarbeiteten Lösung. Bei dem Suchbeispiel stellt es das Auswerten der Suchergebnisse und Auswahl des am passendsten Ergebnisses dar. Weiters wird ausgeführt, dass wenn ein Wechsel zur Unterbrechung erfolgt – im Fall von [14] Sofortnachricht eines Messaging-Systems – dauert dies in der Ausführungs-Phase länger als in der Planungs- oder Evaluierungs-Phase dauert. Außerdem ist eine Sofortnachricht, wenn der Benutzer gerade tippt oder mit einer Toolbar interagiert, störender als zu einem anderen Zeitpunkt. Überdies wurde festgestellt, dass eine Wiederaufnahme der Aufgabe bei nicht (für die Aufgabe) relevanten Unterbrechungen länger dauert, als bei relevanten Unterbrechungen.

		Informationswahrnehmung	
		Niedrig	**Hoch**
Störungseinfluss	**Niedrig**	Hintergrund-Fenster	**Optimal**
	Hoch	Blinkende Systembenachrichtigungen (Infobereich der Taskleiste)	Dialog-Fenster

Tabelle 2.1: Zeigt gängige Methoden, welche in webbasierten *UIs* Verwendung finden und stellt den Grad der Störung mit jenem der Informationswahrnehmung gegenüber [6].

Neben der Minimierung der Störung des Benutzers durch einen *Interrupt*, sind auch Hilfestellungen für die Wiederaufnahme der unterbrochenen Aufgabe sinnvoll. So ist es vor allem beim Durchsuchen von Listen hilfreich, wenn die letzte Position vor der Unterbrechung sichtbar ist, um schnell weitersuchen zu können. Bei einer Untersuchung [14] wurde eine Liste mit Hilfe der Pfeiltasten durchsucht, wodurch der Cursor auf der letzten Position blieb, als der *Interrupt* auftrat. Dies verkürzte die Wiederaufnahme der Suche nach dem *Interrupt*, verglichen ohne Anzeige der letzten Cursor-Position.

Eine weitere Hilfestellung ist die Möglichkeit, wie in [2] beschrieben, dem Benutzer einen Hinweis zu geben, dass er in Kürze unterbrochen wird. So kann der Benutzer sich noch auf seine Hauptaufgabe konzentrieren, was zu einer besseren Wiederaufnahme der unterbrochenen Aufgabe führt. Es wurde festgestellt, dass während eines längeren *Interrupts* (> 6 Sekunden) der Bereich des Bildschirmes in dem zuvor der Benutzer gearbeitet hat, sichtbar bleibt, die Wiederaufnahme schneller erfolgt.

In [3] wird ebenfalls festgestellt, dass, wenn bei einer Unterbrechung ein entsprechender Hinweis gegeben wird, die Wiederaufnahme besser/schneller ist, als ohne. Weiters wird aufgezeigt, dass die Art des Hinweises (visuell oder auditiv) keine wesentliche Rolle spielt. Viel wichtiger ist, dass genügend Zeit bleibt, um dies zu verarbeiten und das Ziel der Aufgabe ins Gedächtnis zu rufen.

Neben den gängigen Benutzer-Benachrichtigungen visuell oder auditiv, gibt es auch Untersuchungen zu Alternativen wie Hitze, Geruch, Vibration, Geräusch und Licht, siehe [4, 5]. So wird in [4] festgestellt, dass Geruch, Vibration und Ton von den Probanden als störender eingestuft wurden als Hitze und Licht. Es wurde jedoch festgehalten, dass auch der Werdegang (Beruf) als auch die Weltanschauung der Probanden eine Rolle spielt, wie sie im Einzelnen auf das alternative Interface reagierten.

In der weiterführenden Studie [5] wurde Hitze und Licht miteinander verglichen. Hierbei zeigt sich, dass Hitze schneller bemerkt wird als Licht aber gleichzeitig auch schwieriger zu ignorieren ist. So ist ein Fertigstellen der derzeitigen Aktivität bei einem ungünstigen Zeitpunkt der Unterbrechung kaum möglich. Außerdem wurde Hitze oft mit Gefahr (Brand) verbunden. Anders hingegen bei Licht: Es wird nicht immer gleich registriert und man kann es auch eine Zeit ignorieren.

Es besteht auch ein Zusammenhang zwischen Komplexität der Aufgabe und der Störung der Unterbrechung [11, 12]. So ist eine Unterbrechung während einer Reihe von zusammenhängenden Abläufen mehr störend. Ein Beispiel hierfür wäre das Verschieben eines Absatzes. Um einen Absatz verschieben zu können, muss der Absatz ausgeschnitten, die neue Stelle gesucht und schließlich der Absatz wieder eingefügt werden. Erfolgt eine Unterbrechung während der Suche nach der neuen Stelle, so ist es für den Benutzer schwieriger – ähnlich bei einer Listensuche wie oben beschrieben – sich wieder im Dokument zu orientieren. Wodurch die Probanden länger zur Bearbeitung der Aufgabe benötigen als ohne Unterbrechung.

So wurde auch bei [1] festgestellt, dass bei einem vorhergesagten ungünstigen Zeitpunkt die Unterbrechung zu mehr Frustration und höherem Störfaktor führt. Zur Feststellung eines günstigen bzw. ungünstigen Zeitpunkts, wurde die Aufgabe in mehrere Teilbereiche aufgeteilt. Als günstig wurde ein Zeitpunkt festgelegt, wenn zwei Teilbereiche, kontextbezogen, voneinander

getrennt werden können. Besonders ungünstig sind Unterbrechungen während man tippt, mit einer Toolbar interagiert oder in der Menüleiste arbeitet.

Auch die Ergebnisse von [7] zeigen, dass der Grad der Störung steigt, wenn man während der Hauptaufgabe oder gleich nach dem Fertigstellen der Hauptaufgabe unterbrochen wird. Neben dem Zeitpunkt ist auch die Art der Hauptaufgabe ausschlaggebend für den Störfaktor. Weiters geben Probanden an, dass Aufgaben, welche unterbrochen werden, schwieriger fertigzustellen sind, als ohne Unterbrechung. Außerdem wird deutlich, dass eine Unterbrechung zu einer längeren Bearbeitungszeit führt.

Gegen die Ergebnisse, dass durch eine Unterbrechung auch zwangsläufig der Zeitaufwand steigt, spricht eine Untersuchung von Mark et al. [24]. Sie haben festgestellt, dass jede Unterbrechung zu einer Veränderung der Arbeitsweise führt. Messungen zeigen, dass unterbrochene Arbeiten schneller abgearbeitet werden. Sie führen dies darauf zurück, dass die Probanden die durch die Unterbrechung verloren gegangene Zeit wieder einarbeiten wollen. Diese Arbeitsweise führt jedoch zu größerem Zeitdruck, Stress, höherer Arbeitsbelastung, mehr Frustration und Arbeitsaufwand. Es wurde festgestellt, dass die Zeit vor allem durch die Neuorientierung nach der Unterbrechung verloren geht.

In [13] wird aufgezeigt, dass die Neuorientierung nach einer Unterbrechung verkürzt werden kann, in dem man diese Phase trainiert. Es tritt nicht nur bei Wiederholung der Hauptaufgabe ein Lerneffekt ein, sondern auch bei der Wiederaufnahme nach einer Unterbrechung. Wiederholt man einen Arbeitsprozess öfter und wird dabei unterbrochen, kann der Zeitverlust bis zum Weiterarbeiten reduziert, aber nicht völlig eliminiert werden.

Durch die Auswirkung von Unterbrechungen auf den Arbeitsprozess und Arbeitsweise, als auch der Zeitpunkt an dem die Unterbrechung auftritt, wird in [7, 24] ein „Attention-Management-System“ vorgeschlagen. Dieses System, soll den Zeitpunkt der Unterbrechungen regulieren, wenn es beim Auftreten des *Interrupts* ungünstig ist. Dadurch sollen die negativen Effekte (Frust, Stress etc.) reduziert werden und auch den Benutzer vor Überlastung schützen. Ein derartiges System ist in der Ausführung sehr komplex und wird in dieser Arbeit nicht weiterverfolgt.

2.2 Veränderung des Sitzverhaltens

Bei *Breakaway* [20] wird untersucht, wie ein Proband auf ein *Ambient Display* reagiert. Bei diesem Versuch mit nur einer Testperson wird mit Hilfe von Sensoren gemessen, wie lange der Proband bereits sitzt. Nach einer Stunde wird ein Motor angesteuert, der das *Ambient Display* dazu bringt, seine Form zu ändern. Die Änderung der Form hat den Probanden dazu aufgerufen, eine Pause einzulegen und aufzustehen. Im anschließenden Interview an

den Versuch wird einerseits deutlich, dass schon ein bis zwei Formänderungen den Benutzer auf sein zu langes Sitzen aufmerksam machten. Andererseits sind diese Formänderungen nicht so störend wie eine Kalendererinnerung und kann im Bedarfsfall (zusammenhängender Arbeitsprozess etc.) auch ignoriert werden, bis der Zeitpunkt günstiger ist.

Ein vergleichbarer Versuch [16], jedoch mit sechs Probanden zeigt ein ähnliches Ergebnis. Es wird ein entsprechender Testaufbau verwendet: Bürosessel mit zwei Sensoren – einer in der Rückenlehne, der andere in der Sitzfläche – und einem *Ambient Display* hier *Agent* genannt. Bei der Laborstudie sinkt die Zahl an schlechten Sitzpositionen, wenn der *Agent* verwendet wird. Bei diesem Versuch wird als richtiges Sitzen angesehen, wenn auf beide Sensoren (Rückenlehne und Sitzfläche) ein Druck ausgeübt wird. Dies wiederum bedeutet eine einseitige Belastung beim Sitzen und entspricht nicht mehr aktuellen Studien über gesundes Sitzen.

Kapitel 3

Konzept

In diesem Kapitel wird das grundlegende Konzept für das System erläutert, welches das Sitzverhalten des Benutzers erfasst und überprüft. Es werden die dafür nötigen Komponenten aufgeführt und erklärt.

3.1 Allgemeines

Die in Abschnitt 1.2 beschriebenen gesundheitlichen Folgen von schlechtem Sitzverhalten sollen vermieden werden, indem der Benutzer auf seine Sitzhaltung aufmerksam gemacht wird. Dies wird erreicht indem ein handelsüblicher Bürosessel mit entsprechendem Messequipment ausgestattet wird[18], eine Software die gemessenen Daten auswertet und über die Sitzhaltung informiert. Ziel des resultierenden Prototypen (Sessel + Software) ist es, den Benutzer zu einem dynamischen Sitzverhalten (siehe Abschnitt 1.1) zu bewegen. Der Prototyp soll für eine Langzeitstudie zum Einsatz kommen, in der die Wirkung des Prototypen getestet wird. Zur Feststellung der Wirksamkeit wird auch ein *Swopper* (siehe Abschnitt 1.2) mit der entsprechenden Messtechnik ausgestattet. Dadurch soll ein Vergleich des dynamischen Sitzens auf dem *Swopper* mit jenem des Prototypen ermöglicht werden.

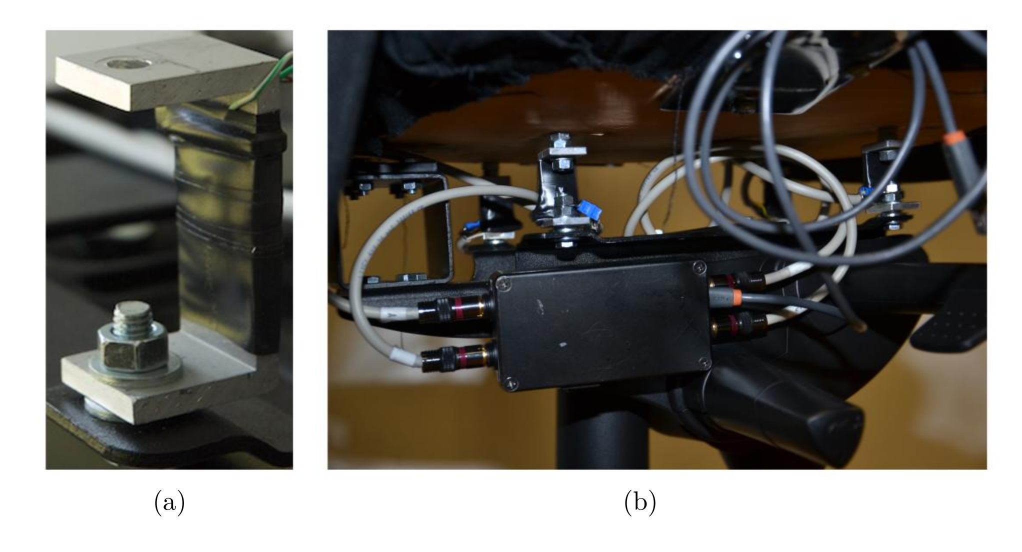
3.2 Sensoren

Um Messdaten über die Position einer sitzenden Person zu erhalten, wird ein Aluminium C-Profil [19] mit Dehnungsmessstreifen (DMS) ausgestattet − siehe Abbildung 3.1 (a). DMS sind dünne Folien mit einem Messgitter, dessen elektrischer Widerstand sich bei Dehnung verändert.

---

[18] Die Ausstattung der Sessel mit der Messtechnik erfolgte an der Fachhochschule Linz (Bereich Medizintechnik).

[19] EN AW-6060, AlMgSi0.5. Maße: 40 × 40 × 40 × 4mm Wandstärke.



(a) (b)

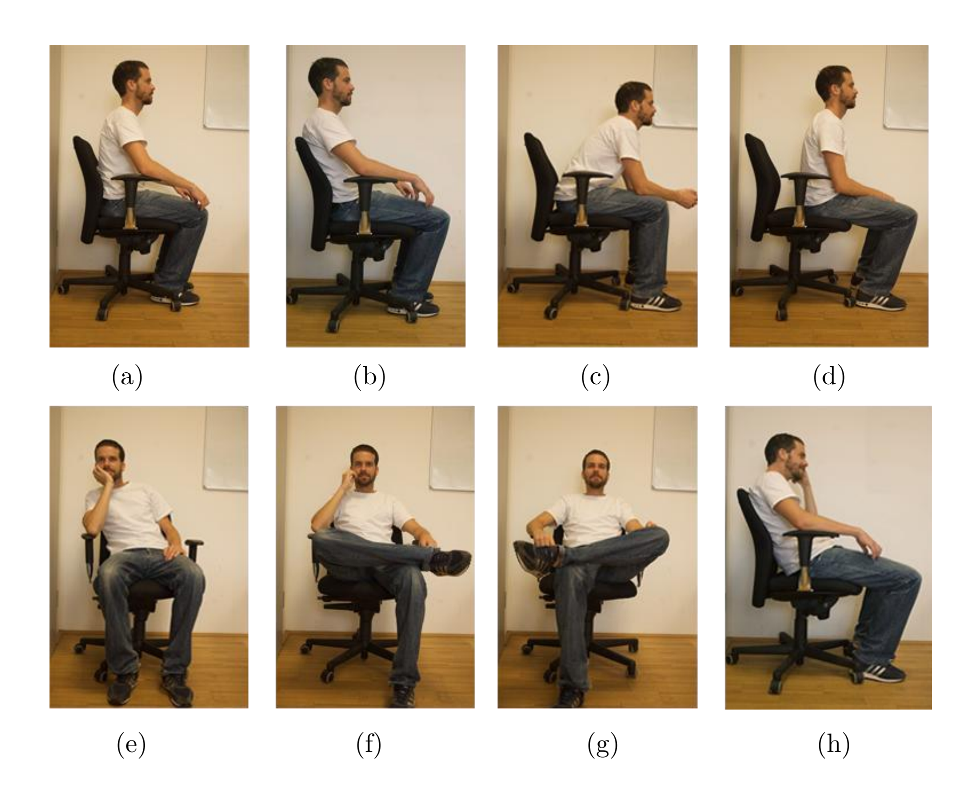
Abbildung 3.1: C-Form Aluminium mit Dehnungsmessstreifen(a), Verbindung der DMS mit dem *ACAM Modul*(b).

Jeder Sensor ist mittels eines Kabels mit dem *ACAM Modul PS021* [20] verbunden – siehe Abbildung 3.1, b – welches die Messwerte (elektrischer Widerstand) digitalisiert und über ein *Serial Peripheral Interface* an die Sensorplattform *NEON* [21] sendet. Am *NEON* wird aufgrund der Messdaten aller Sensoren der *Center of Pressure (COP)* als X- und Y-Koordinate berechnet. Die *COP*-Daten werden mittels dem kabellosen Protokoll *ANT* [22] an das entsprechende Empfängermodul (*ANT* USB Stick) am PC gesendet.

3.3 Intelligenter Bürosessel

Für das Projekt wurde die Sitzfläche handelsüblicher Bürosesseln, mit vier Sensoren – siehe Abschnitt 3.2 – ausgestattet. Die Sensoren wurden in bereits vorhandenen Bohrungen des Herstellers angebracht. Durch die Sensoren kann die aktuelle Sitzposition und Haltung der sitzenden Person eruiert werden. Abbildung 3.2 zeigt mögliche Positionen während eines Arbeitstages, die mit dem intelligenten Bürosessel gemessen werden können [26]. Die Sitzposition und Haltung wird ca. 15-mal pro Sekunde erfasst und kann drahtlos an ein entsprechendes Empfängermodul am Computer übertragen werden. Durch das Anbringen der Sensoren stimmt das Höhenverhältnis zwischen Sitzfläche

[20] http://www.acam.de/products/picostrain/psoe21/
[21] http://www.spantec.at/sensortechnik/neon/
[22] http://www.thisisant.com/pages/developer-zone/ant-protocol-and-usage



(a) (b) (c) (d)

(e) (f) (g) (h)

Abbildung 3.2: Mögliche Sitzhaltungen, die mit dem intelligenten Bürosessel gemessen werden können. (a) aufrechte Sitzhaltung, (b) zurückgelehnte Haltung, (c) nach vorne gelehnt, (d) Sitzen auf der Vorderkante des Sessels, (e) nach rechts gelehnte Sitzhaltung, (f) rechtes Bein über dem Linken, (g) linkes Bein über dem Rechten, (h) klassisches Herumlungern. Quelle: Schrempf et. al [26].

und Rückenlehne nicht mehr überein. Deshalb wurde die Höhe und Position mit Hilfe eines Zwischenstückes ungefähr wiederhergestellt, damit ein angenehmes Sitzen gewährleistet ist. Während der Studie wurde die Höhe und Position der Rückenlehne in dieser Einstellung belassen, da das Verstellen der Lehne die Messergebnisse beeinflussen könnte. Insgesamt wurden acht Bürosessel mit Sensoren bestückt.

3.4 Swopper

Um einen Vergleich zu aktiven Sitzgelegenheiten zu erhalten, wurden auch vier *Swopper* mit Sensoren ausgestattet, welche von *aeris*[23] zur Verfügung gestellt wurden. Die Sensoren sind im inneren des *Swoppers* angebracht – siehe Abbildung 3.3.

[23] http://www.swopper.de

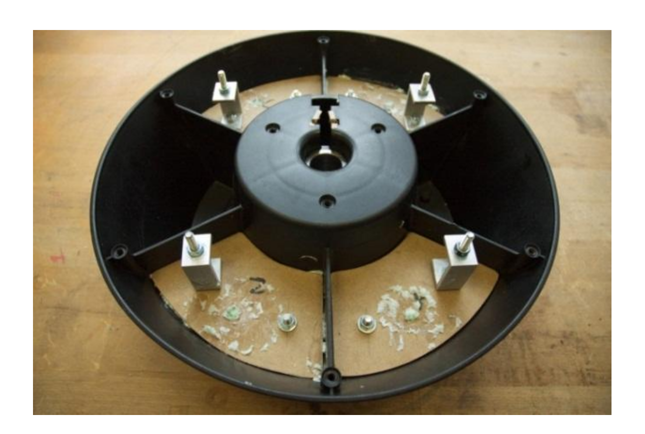
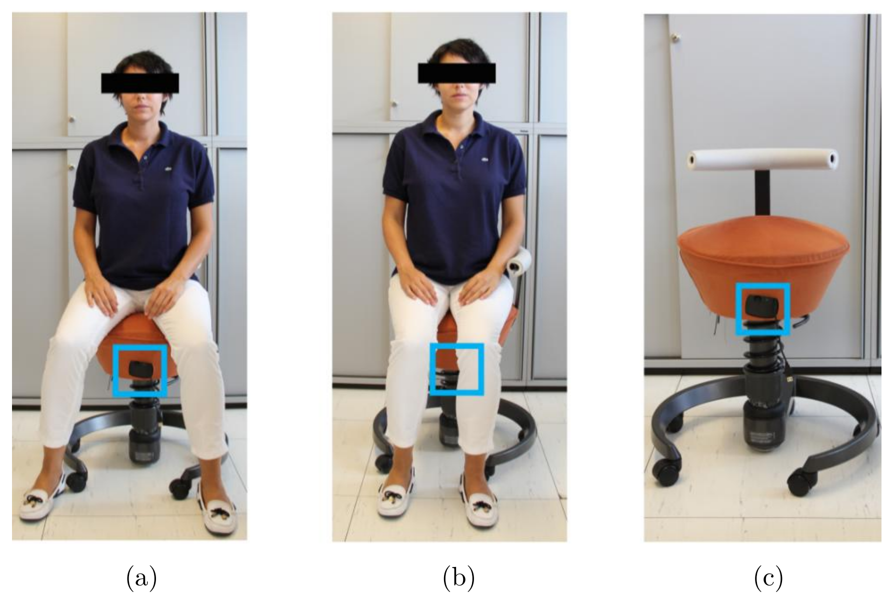


Abbildung 3.3: Innenansicht der Sitzfläche eines *Swoppers* mit den Sensoren.

Durch die freidrehende, runde Sitzfläche des *Swoppers* gibt es kein links, rechts bzw. kein vorne oder hinten. Dies ist jedoch für die Messergebnisse von entscheidender Bedeutung, da die Berechnung der *COP* auf Basis der Anordnung der Sensoren erfolgt. Nimmt ein Proband auf dem *Swopper* Platz und ist die Ausrichtung der Sensoren wie in Abbildung 3.4 (a), erfolgt die Messung und Auswertung richtig. Hingegeben bei (b) der Abbildung 3.4 erhält man verfälschte Messergebnisse. Um falschen Messergebnissen vorzubeugen, wurde am *Swopper* eine Rückenlehne montiert – siehe Abbildung 3.4 (c).



(a) (b) (c)

Abbildung 3.4: Richtige Anordnung der Sensoren während des Sitzens (a); falsche Anordnung der Sensoren (b). *Swopper* mit Rückenlehne (c).

Durch die Rückenlehne ist gewährleistet, dass der Proband immer Richtig am Sessel Platz nimmt. Trotz Rückenlehne bleibt die Bewegungsfreiheit (freies Drehen und Gegenbewegung durch *Swopper*) und viel wichtiger das aktive Sitzen gewährleistet.

3.5 Kostenfunktion

Schrempf et al. haben in [26] die Kostenfunktion zur Bestimmung der Sitzposition erörtert. Durch

$$J_p = \frac{1}{t_N - t_1} \sum_{i \,\epsilon\, \{x,\, y\}} \left(C_1 \, \|\varphi_i\|^2 + C_2 \, e^{-C_3 \left(\|\Delta \overline{COP}\| \right)^2} \right) = J_{p\varphi} + J_{pM}$$

kann die Sitzposition einer, auf dem intelligenten Bürosessel sitzenden, Person errechnet werden. Die Formel ist in einen statischen $J_{p\varphi}$ und einen dynamischen Teil J_{pM} gegliedert und wird über einen Zeitraum *t* berechnet.

Der statische Teil gibt Aufschluss über die Haltung der Wirbelsäule. So ist eine aufrechte Haltung der Wirbelsäule besser, als eine nach vorn, hinten oder seitlich geneigte Haltung. Bei der aufrechten Haltung ist das Moment, welcher auf die einzelnen Wirbel wirkt geringer und somit auch schonender.

Der dynamische Teil der Kostenfunktion bewertet die „Bewegung“. Um ein dynamisches Sitzen zu erfassen, wird unter anderem der gleitende Mittelwert über die (im Zeitraum *t*) gesammelten *COP*-Werte berechnet. Somit wird verhindert, dass eine kurzfristige, große Positionsänderung (z.B. nach links oder rechts) zu großen Einfluss auf die gesamte Sitzposition im Zeitraum *t* hat.

Die Konstanten C_1, C_2 und C_3 sind von der Fachhochschule Linz anhand von empirischen Messungen bestimmt worden. Bei der Ermittlung der Konstanten wurde darauf geachtet, dass je nach Sitzverhalten der dynamische oder der statische Teil der Kostenfunktion mehr Einfluss auf das errechnete Ergebnis hat. Damit dies der Fall ist, müssen die Konstanten mit Werten belegt werden, die ausgewogene Teilergebnisse zulassen. So wurden die Konstanten C_1 und C_2 mit 1 belegt und C_3 mit 15. Durch diese Belegung der Konstanten, wird ein Ergebnis erzielt, dass mit Schwellwerten verglichen und somit eine Bewertung (gut/dynamisch, neutral, schlecht/statisch) der Sitzposition erfolgen kann. Die Schwellwerte wurden ebenfalls anhand von Messungen der Fachhochschule Linz eruiert.

Kapitel 4

Studie

Dieses Kapitel erörtert das Finden des besten Ansatzes für die Rückmeldung über das Sitzverhalten. Die Ermittlung der geeignetsten Rückmeldung erfolgt anhand einer Studie mit unterschiedlichen Rückmeldesystemen.

4.1 Studiendesign

Die Vorstudie [19] diente vor allem dazu, die Rückmeldungsvarianten, welche auf Basis der *Interrupt*-Recherche – siehe Abschnitt 2.1 – als Testanwendung implementiert wurden, auf ihren Störungsgrad und ihre Akzeptanz bei Benutzern zu überprüfen. Es wurden drei verschiedene Rückmeldungen mit zwölf Probanden getestet. Um den Störungsgrad festzustellen, wurden den Probanden verschiedene Aufgaben gestellt und verschiedene Daten während der Bearbeitung der Aufgabe erhoben, sowie eine Befragung der Probanden durchgeführt. Die Probanden mussten folgende Aufgaben bearbeiten:

- Editieren eines Textdokumentes,
- Schreiben einer Transkription eines Videos und
- Das Suchen und Planen einer Reise.

Während des Bearbeitens der einzelnen Aufgaben wurde jeweils eine Rückmeldung aktiviert. Alle Probanden bewältigten alle drei Aufgaben mit allen drei Rückmelde-Varianten in unterschiedlicher Reihenfolge. Für die einzelnen Aufgaben standen den Probanden jeweils zehn Minuten zur Verfügung.

Die Information über das Sitzverhalten wurde nicht auf Basis von Messdaten vom intelligenten Bürosessel berechnet, sondern innerhalb einer vorgegebenen Zeitspanne generiert. Dadurch wurde verhindert, dass ein Proband, aufgrund seiner guten Sitzhaltung, nie eine Rückmeldung erhalten würde.

4.2 Rückmeldung

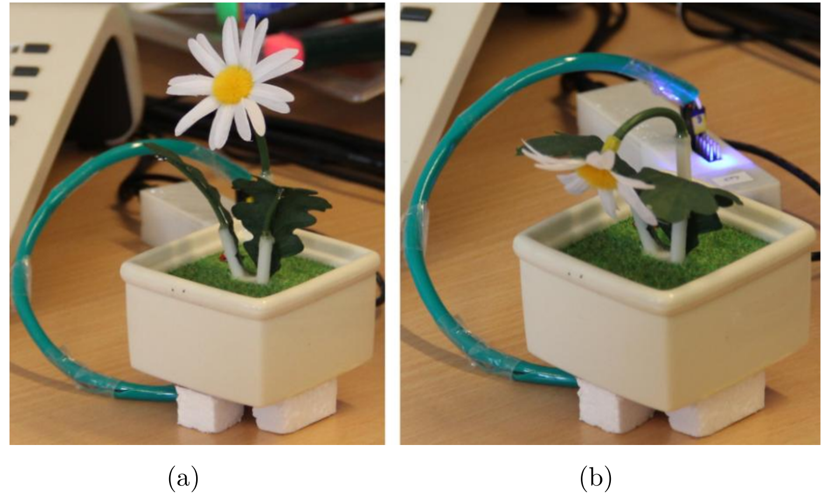
Auf Basis der in Abschnitt 2.1 erörterten *Interrupt*-Eigenschaften, wurden drei verschiedene Lösungsansätze [19] erarbeitet, um den Benutzer auf seine Sitzhaltung aufmerksam zu machen:

- Graphische Rückmeldung,
- Physische Rückmeldung und
- Vibrierende Rückmeldung.

Bei der graphischen Rückmeldung ändert sich die Farbe des Icons in der Taskleiste, um die Sitzposition anzuzeigen. Die Farben wurden von grün (für sehr gut) bis dunkelrot (für sehr schlecht) abgestuft.

Die physische Rückmeldung erfolgt über eine in Abbildung 4.1 dargestellte, modifizierte *Hanappa*-Blume. Die *Hanappa* wurde dahingehend modifiziert, dass man über *Universal Serial Bus (USB)* die Blätter bzw. die Blüte, in einzelnen Schritten, um bis zu 60° neigen kann. Diese schrittweise Neigung wird durch *Shape Memory Alloy (SMA)* erreicht. *SMA* kann man durch Hitze (in diesem Fall Hitze durch Strom) in beliebige Formen bringen. Wird die Stromzufuhr verringert bzw. völlig gestoppt, nimmt das *SMA* seine ursprüngliche Form wieder an, da es sich an den Ausgangszustand „erinnert".

Die Vibrationsrückmeldung wurde durch einen handelsüblichen Gamecontroller mit *force feedback*[24] erzeugt, welcher an die Unterseite des Büro-



(a) (b)

Abbildung 4.1: *Hanappa* mit aufrechter Blüte für sehr gute Haltung (a). *Hanappa* mit maximal geneigter Blüte für sehr schlechte Haltung (b).

[24] Der Kontroller kann eine haptische Rückmeldung durch Vibration erzeugen.



Abbildung 4.2: Der am Bürosessel montierte Gamecontroller (rot eingerahmt).

sessels montiert wurde – siehe Abbildung 4.2. Durch das verwendete *Logitech Rumble-pad 2*[25] konnten unterschiedlich starke Vibrationsstufen eingestellt werden.

4.3 Training

Neben der Information über die aktuelle Sitzposition wird auch das Sitzverhalten über einen längeren Zeitraum festgehalten. Sitz man zu lange statisch, können – wie in Kapitel 1 beschrieben – verschiedenste Probleme mit Muskelpartien und der Wirbelsäule auftreten. Um dies zu verhindern, wird nach einer vordefinierten Zeitspanne, der Benutzer zu einer 30 Sekunden dauernden Trainingseinheit aufgefordert. Bei der Vorstudie wurde bei jeder Aufgabe – unabhängig von der Sitzposition - mindestens eine Trainingsaufforderung angezeigt. Die Aufforderung selbst wird je nach eingestellter Rückmeldung anders angezeigt. Bei der graphischen Rückmeldung erscheint die Aufforderung wie das *Adjusting Window* von Bailey et al. [6]. Das *Adjusting Window* verkleinert im Browser das Hauptfenster, um einen vordefinierten Bereich, um den Benutzer auf neue Meldungen aufmerksam zu machen. Im Unterschied zu Baileys Ansatz, der sich auf den Browser beschränkt, wurde für die Vorstudie eine globale Lösung geschaffen, damit jedes Fenster des Desktops um den gewünschten Bereich verkleinert wird. Bei der physischen bzw.

[25] http://www.log-itech.com/de-de/gaming/controllers/devices/264

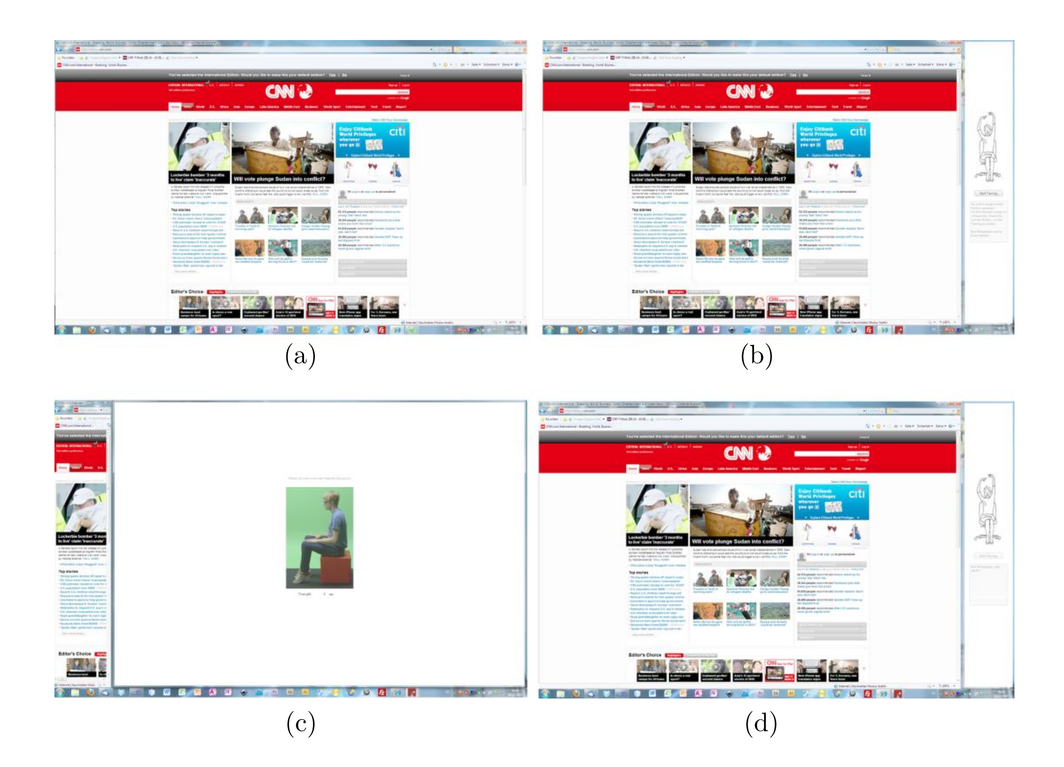


Abbildung 4.3: *Adjusting Window* Testanwendung der graphischen Rückmeldung. Vor der Trainingsaufforderung (a), Aufforderung zum Training (b), Abspielen des Trainingsvideo nach der Größenänderung der Fenster (c), Nach Beendigung der Trainingseinheit und der Wiederherstellung der Fenstergrößen.

vibrierenden Rückmeldung wird ein neues Fenster minimiert geöffnet. Das minimierte Fenster blinkt in der Taskleiste. Während der Aufforderung bewegt sich die *Hanappa* auf und ab um ebenfalls auf die Aufforderung aufmerksam zu machen bzw. werden auch die einzelnen Vibrationsstufen durchgeschaltet. Der Inhalt der Aufforderung ist identisch gleich der Art der Rückmeldung, nämlich eine kurze Erläuterung zur Aufforderung, sowie ein Start- bzw. ein Minimieren-Button. Wird der Start-Button angeklickt so wird bei der graphischen Rückmeldung die Größe des Aufforderungsfensters auf jene Größe des derzeit aktiven Fensters geändert und ein Video abgespielt, mit der jeweils zu absolvierenden Übung, wie Abbildung 4.3 zeigt. Das aktive Fenster wird auf den verbleibenden Platz am Desktop verkleinert. Ist das Trainingsvideo zu Ende, werden die Fenstergrößen wie vor dem Training wiederhergestellt – siehe Abbildung 4.3. Bei der physischen bzw. vibrierenden Rückmeldung wird lediglich der Inhalt des Aufforderungsfensters gegen das Trainingsvideo ausgetauscht.

4.4 Ergebnisse

Die wichtigsten Erkenntnisse aus der Studie werden hier zusammengefasst. Alle Ergebnisse und Daten die erhoben wurden sind detailliert in Haller et al. [19] aufgelistet.

Die *Hanappa* wurde von den Probanden sowohl als Aufforderung zum Training als auch bei der fortlaufenden Rückmeldung bevorzugt, da sie als weniger störend empfunden wurde als die beiden anderen Methoden. Die Vibration wurde als am Störendsten eingestuft und die Probanden gaben an, dass sie die Vibration auf einen längeren Zeitraum vermutlich abgestellt hätten, da sie *überaus störend* sei.
Manche Probanden gaben an, dass sie das Icon der graphischen Rückmeldung nicht wahrgenommen haben, während andere durchaus bewusst auf dieses geachtet haben. Die Auswertung der Daten der Augenbewegung [26] zeigte jedoch, dass alle Probanden auf das Icon geblickt haben. Dies lässt darauf schließen, dass sie, während sie zum Beispiel zwischen zwei Fenster in der Taskleiste wechselten, kurz auf das Icon sahen, es aber nicht bewusst wahrgenommen haben.

Bei der physischen Rückmeldung durch die *Hanappa*- bzw. bei der Vibrationsrückmeldung konnte eine sehr schnelle Wiederaufnahme der gestellten Aufgabe festgestellt werden. Die Auswertung der Augenbewegungen weist bei sieben Probanden eine mögliche Erklärung für das schnelle Weiterarbeiten

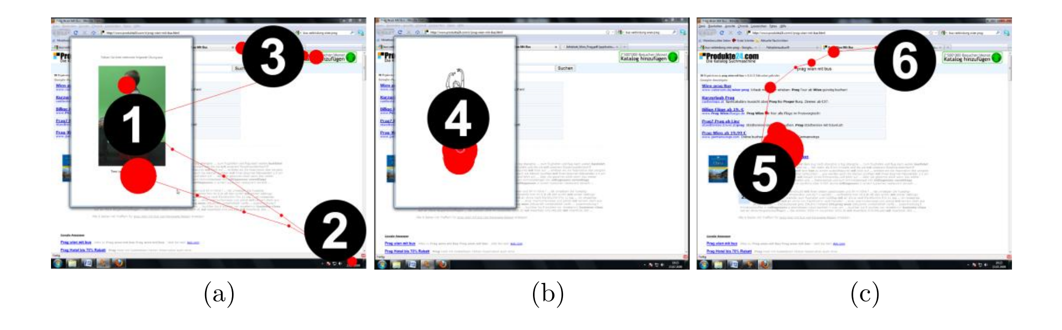


Abbildung 4.4: Augenbewegung der Probanden während einer Übungseinheit [19]. Der Proband beobachtet die auf dem Video ersichtliche Übung (1) und überprüft die aktuelle Uhrzeit in der Taskleiste (2). Danach überlegt er sich weitere Schritte, um die gestellte Aufgabe zu erfüllen (3). Nach dem Beenden der Übungseinheit ist der Blick auf die *Schließen* Schaltfläche (4) gerichtet. Nach dem Schließen des Trainingsfensters (5), wird wie geplant auf den entsprechenden Tab im Browser geklickt (6) – siehe Haller et al. [19]

[26] Die Augenbewegung wurde mit Hilfe des *Tobii Eye Tracker* aufgezeichnet. http://www.tobii.com/en/eye-tracking-research/global/products/hardware/tobii-t60t120-eye-tracker/

auf. Da bei diesen Varianten der Rückmeldung, der Arbeitsbereich durch das Trainingsfenster minimal verdeckt wird[27], konnte der Benutzer weitere Schritte zur Bearbeitung der Aufgabe vorausplanen. Wie Abbildung 4.4 zeigt, plante der Benutzer die nächsten Aktionen im Browser voraus, während er die geforderte Übung absolvierte.

[27] Im Vergleich zur graphischen Rückmeldung.

Kapitel 5

Redesign

Dieses Kapitel beschreibt die Überarbeitung des Konzeptes des Prototypen für eine Langzeitstudie. Neben dem vorgestellten Konzept aus Kapitel 3 fließen auch die Ergebnisse der Studie aus dem vorherigen Kapitel ein. Des Weiteren muss der Prototyp gewisse Anforderungen erfüllen, damit die Langzeitstudie durchführbar ist.

5.1 Prototyp für Langzeitstudie

Die Ergebnisse der Vorstudie aus Kapitel 4 wurden herangezogen um die bestehende Konzeptionierung zu überarbeiten und einen lauffähigen Prototypen zu erhalten. Der Prototyp wird bei einer Feldstudie [28] unter realen Bedingungen getestet. Die *Hanappa*, sowie die graphische Rückmeldung schnitten bei der Vorstudie besser ab, als das Vibrationsfeedback. Die graphische Rückmeldung wurde überarbeitet, da sie nicht ganz so gut abgeschnitten hat wie die *Hanappa*. Beide Rückmeldesysteme werden für die Feldstudie eingesetzt und erneut gegenübergestellt.

5.2 Verbesserte graphische Rückmeldung

Das plötzliche Verkleinern des aktiven Fensters bei der Trainingsaufforderung wirkte störend, weshalb dieser Teil der Rückmeldung neu überdacht wurde. Das Prinzip des *Adjusting Windows* wurde beibehalten, jedoch neu konzipiert. Das verbesserte Konzept verkleinert nicht die Größe des Hauptfensters, wodurch hier keinerlei Einfluss auf die Verwendung von Menüs (falsche Auswahl durch Verschiebung der Menüposition) oder Textumbrüche genommen wird. Anstatt der Verkleinerung, wird die Aufforderung seitlich, über das

[28] Die Datenauswertung der Langzeitstudie ist nicht Teil dieser Arbeit.



Abbildung 5.1: Verbesserte Rückmeldung. Vor der Trainingsaufforderung (a), Aufforderung zum Training (b), Abspielen des Trainingsvideo nach der Expansion des Fensters (c), Nach Beendigung der Trainingseinheit und der Breitenminimierung des Fensters.

Hauptfenster geschoben, siehe Abbildung 3.9 (b). Um so wenig wie möglich vom Hauptfenster zu verdecken, besitzt das Aufforderungsfenster eine maximale Breite und ist größtenteils transparent. Wird das Training gestartet, so dehnt sich das Fenster über die gesamte Bildschirmbreite aus, wie Abbildung 5.1 (c) zeigt. Das soll den Fokus als erstes auf das Trainingsvideo in der Mitte ziehen. Da auch hier der Bereich um das Video transparent ist, kann der Benutzer während der Übung schon die nächsten Schritte planen, was sich bei der Vorstudie als vorteilhaft herausgestellt hat – siehe Abbildung 4.4. Diese Änderungen sollen den Ergebnissen der Vorstudie gerecht werden und eine vergleichsweise Handhabung wie die Rückmeldung der *Hanappa* bieten.

5.3 Training

Die vier Trainingsvideos basieren auf vorgeschlagene Übungen von *Kuhnt's Rückenschule* [22] – siehe Abbildung 5.2. Es wurden vier Übungen ausgewählt, welche sich über die Sensoren klar messen und die einzelnen Bewegungen der Übung unterscheiden lassen.

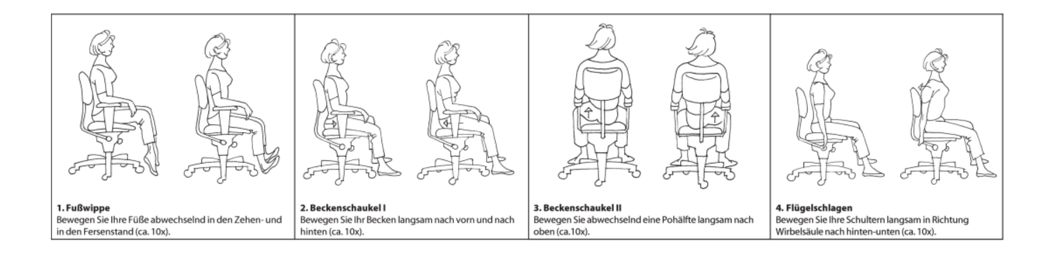


Abbildung 5.2: graphische Darstellungen [22] der zu absolvierenden Übungen im Trainingsmodus. Anhand dieser Grafiken wurden die Trainingsvideos gedreht.

Als maximalen Zeitraum für statisches Sitzen werden die von Ludwig et al. [23] definierten 25 Minuten herangezogen. Für eine regelmäßige Rückmeldung an den Benutzer werden die Messdaten alle drei Minuten ausgewertet. Ergibt die Auswertung zum siebten Mal in Folge eine statische Sitzhaltung, das einem Zeitraum von 21 Minuten entspricht, so wird eine Trainingsaufforderung angezeigt.

5.4 Anforderungen

Um bei der Feldstudie eingesetzt werden zu können, muss der Prototyp folgende Anforderungen erfüllen:

- Unterstützung von *Windows XP* und *Windows 7*,
- Sammlung von Daten ohne Rückmeldung (als Vergleichsdaten),
- Sammlung von Daten mit Rückmeldung,
- Verwendung der Physische Rückmeldung,
- Verwendung der graphische Rückmeldung,
- Berechnung der Sitzposition mittels Kostenfunktion – siehe Abschnitt 3.5,
- Bewertung der Sitzposition auf Basis von vordefinierten Schwellwerten,
- Bewertung des Trainings auf Basis von vordefinierten Schwellwerten,
- Einfache Einstellmöglichkeiten von Konstanten und Variablen (z.B. Konstanten der Kostenfunktion, Körpergewicht und – größe der Probanden etc.) und
- Datenübertragung mittels *ANT* Protokoll.

Kapitel 6

Implementierung

Auf den folgenden Seiten wird erläutert wie das Konzept und die erhaltenen Ergebnisse der Vorstudie als Programm umgesetzt und gängige Programmierkonzepte angewandt wurden. Der daraus resultierende Prototyp wird für den Einsatz der Langzeitstudie herangezogen.

6.1 Allgemeines

Der Prototyp besteht aus zwei separaten Implementierungen, welche in *Visual Studio*[29] mit der Programmiersprache *C#* umgesetzt sind. Die Basis des Programmes stellt die Logik dar, während die Benutzeroberfläche das eigentliche Programm repräsentiert, da mit ihr der Benutzer interagiert. Um die nötige Flexibilität der Langzeitstudie zu gewährleisten, wurden die dafür nötigen Einstellungen in einem *Extensible Markup Language (XML)* Dokument ausgelagert – siehe Anhang A Programm A.1.

6.2 Logik

Die Logik ist eine *Dynamic Link Library (DLL)* auf Basis des *.Net Frameworks 4* und stellt die notwendigen Methoden über Schnittstellen bereit. Somit kann die Funktionalität in verschiedenen (*.Net* kompatiblen) Programmen eingebaut werden.

6.2.1 Grundlegende Funktionsweise

Die Logik liest die Daten vom *ANT* Empfänger Modul aus und sammelt diese. Ist eine Messperiode verstrichen, werden die gesammelten Daten mit Hilfe

[29] Entwicklungsumgebung von Microsoft.

der Kostenfunktion ausgewertet. Das Ergebnis der Kostenfunktion wird mit den vordefinierten Schwellwerten verglichen und es kann für die Messperiode eine Aussage über das Sitzverhalten getroffen werden. Diese Aussage wird an das Programm, welche die Logik verwendet, weitergeleitet. Die Aufbereitung des Ergebnisses ist dem jeweiligen Programm vorbehalten.

6.2.2 Aufbau

Die Logik besteht aus einem öffentlichen und einem privaten Namensraum, um die Implementierung abzukoppeln und austauschbar zu machen. Der öffentliche Namensraum besteht, wie in Abbildung 6.1 dargestellt, aus verschiedenen Schnittstellen (*Interfaces*, Präfix I) und einer Klasse namens `LogicFactory`. Für die Verwendung der Logik wird das Entwurfsmuster *Factory Method*[30] verwendet, das durch die `LogicFactory` repräsentiert wird und eine Instanz der Logik zurückgibt. Mit dieser Instanz kann die Funktionalität der Logik mit Hilfe der öffentlichen Methoden der Schnittstelle – Abbildung 6.1 (b) – gesteuert werden. Die strikte Einschränkung durch die Schnittstelle stellt sicher, dass keine andere (private) Methode aufgerufen und der Programmablauf verändert wird.

Die Weitergabe der Daten basiert auf einem *Ereignis-* oder *Eventsystem*.

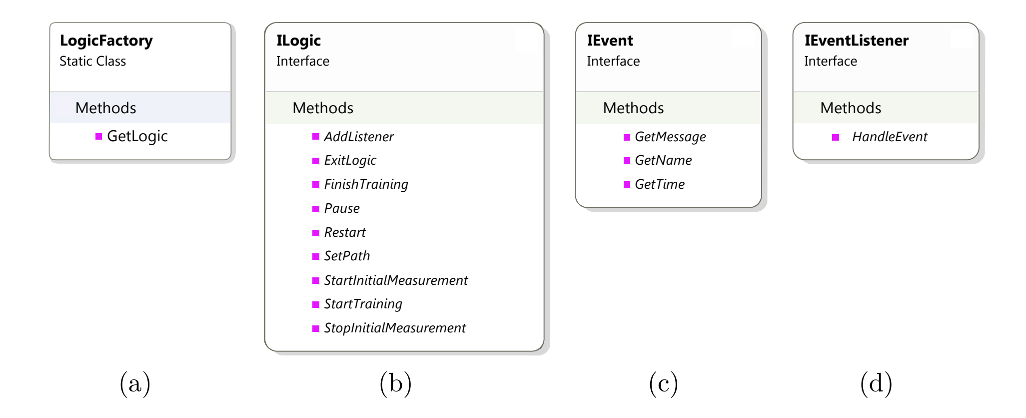


Abbildung 6.1: Zeigt den öffentlichen Namensraum. (a) *LogicFactory*, welche die Implementierung der *Factory Method* darstellt. (b) Die Logik-Schnittstelle mit den zur Verfügung stehenden Methoden. (c) Die *Event*-Schnittstelle, mit den Methoden, die alle *Events* aufweisen. (c) Das `IEventListener` Interface, mit der Methode, welche vom Programm das die DLL verwendet, implementiert werden muss.

[30] Eine schematische Beschreibung zur Lösung von immer wiederkehrenden Problemen in der Informatik.

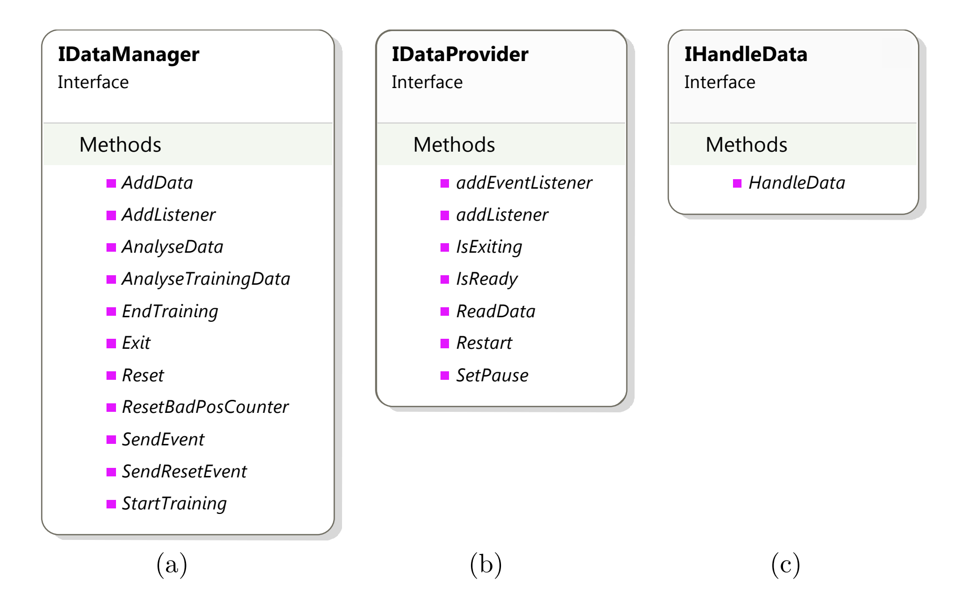


Abbildung 6.2: Schnittstellen im privaten Namensraum. (a) Festgelegte Methoden der `DataManager` Klasse, welche für die Datenverwaltung zuständig ist. (b) Methoden des `DataProvider`, der für den Datenempfang verantwortlich ist. (c) Methode, welche die internen *Listener* zur Ereignisverarbeitung implementieren müssen.

Um *Ereignisse* von der Logik zu erhalten, muss sich das jeweilige Programm bei der Logik als *Listener* registrieren, damit es über Änderung der Daten informiert wird. Gleichzeitig muss das Programm die Methode `HandleEvent(IEvent e)` implementieren, um die Informationen der Logik zu verarbeiten. Der private Namensraum beinhaltet die eigentliche Funktionalität der Logik und ist nicht von anderen Programmen einsehbar. Auch im privaten Namensraum gibt es *Interfaces* um sicherzustellen, dass nur diese Methoden der Klasse von anderen Klassen innerhalb des privaten Namensraumes verwendet werden – siehe Abbildung 6.2. Auch im privaten Namensraum wird ein *Eventsystem* zur Kommunikation zwischen den Klassen verwendet. In Verbindung mit *Threads* ist somit eine verteilte Abarbeitung verschiedenster Funktionen möglich.

6.2.3 Hauptkomponenten

MainLogic

Die `MainLogic` stellt das Bindeglied zwischen den einzelnen Klassen der Logik dar und ist auch für die Datenweitergabe an andere Programme zuständig. Die Klasse ist als *Singleton* implementiert, wodurch es nur eine Instanz

gibt, welche von der `LogicFactory` erzeugt wird. Dadurch können sich mehrere Programme bzw. Klassen eines Programmes bei der Logik für *Events* registrieren und erhalten dieselben Daten und können diese verarbeiten.

Die `MainLogic` beinhaltet Instanzen von der Klasse `DataProvider` (zuständig für den Empfang der Daten) und `DataManager` (verwaltet die empfangenen Daten, Beschreibung im nachstehenden Abschnitt). Die Parameter von diesen Klassen werden zu Beginn von der Logik eingestellt, können aber auch während der Laufzeit verändert werden. Sie selbst, registriert sich bei diesen zwei Klassen um Daten und Änderung zu erhalten und entscheidet, was an die registrierten Programme oder an eine andere Klasse innerhalb der Logik weitergegeben wird. Dadurch sind der `DataProvider` und der `DataManager` voneinander abgekoppelt und können leicht durch andere Implementierungen ersetzt oder erweitert werden, wenn dies erforderlich ist (da in der Testphase des Prototypen durchaus Änderung erforderlich sein können bzw. sich auch die Anforderungen ändern können).

Die `MainLogic` überwacht den aktuellen Status der Datenerfassung. So wird zwischen einem Messmodus, einem Trainingsmodus und einem Lademodus unterschieden. Der Messmodus beschreibt das Erfassen der Daten vom Sessel innerhalb einer Messperiode. Nach jeder Messperiode wird die Sitzhaltung überprüft. Eine Messperiode dauert drei Minuten. Nach sieben Messperioden mit schlechtem Verhalten wird die Trainingsaufforderung aktiviert. Somit ist eine maximale Zeitspanne von 21 Minuten für schlechtes Sitzverhalten veranschlagt und erfüllt somit die in [23] angegebenen Grenzwerte. Wird die Trainingseinheit absolviert, befindet sich die Logik im Trainingsmodus, wodurch die erhaltenen Daten vom Sessel anders verarbeitet werden. Nach Beendigung des Trainingsmodus wird der Messmodus wieder aktiv. Wird der Akku am Sessel aufgeladen, kann der Sessel nicht gleichzeitig die einzelnen Daten der Sensoren auslesen und auswerten, weshalb nur Daten über den Akkustatus gesendet werden. Somit wird der Lademodus aktiv, um die Daten vom Sessel nicht für die Auswertung der Sitzhaltung zu verwenden. Nach dem Aufladen wird wieder in den Messmodus umgeschaltet.

Treten Verbindungs- oder Messprobleme beim Sessel auf, so stellt die `MainLogic` die Methode `Restart()` zum Zurücksetzen der Verbindung bereit. Die MainLogic setzt den Startzeitpunkt, sowie den letzten Messzeitpunkt auf den Zeitpunkt des Neustarts. Weiters werden auch der `DataProvider` und der `DataManager` zurückgesetzt.

Durch das Beenden der Logik durch die `ExitLogic`() Methode, wird die entsprechende Methode des `DataProvider` bzw. des `DataManagers` aufgerufen. Dadurch werden die benötigten Ressourcen wieder freigegen und gesammelte Daten, welche sich noch im Puffer befinden, in die entsprechenden Dateien geschrieben. Sind alle Ressourcen freigegeben, informiert darüber ein *Ereignis*.

DataProvider

Die Klasse `DataProvider` dient dazu, die Daten vom Bürosessel auszulesen und an die `MainLogic` weiterzuleiten. Zu Beginn der Implementierung der Logik stand noch kein intelligenter Bürosessel, der mit *ANT* ausgestattet war, zur Verfügung, da diese erst mit Sensoren ausgestattet werden mussten. Deshalb wurde auch hier das Konzept der *Factory Method* herangezogen. So wird in der Konfigurationsdatei – siehe Anhang A Programm A.1 – angegeben, wie das Einlesen der Daten erfolgen soll. So wurde eine *Dummy-Klasse* implementiert, die das Auslesen der Daten vom Sessel simuliert, sowie eine Klasse, die über das *ANT Protokoll* die Daten vom Sessel ausliest. Somit konnte man die restlichen Funktionen der Logik auch ohne intelligenten Bürosessel testen, aber jederzeit – wenn ein Sessel zur Verfügung stand – auf die Daten vom Sessel zurückgreifen. Auch der `DataProvider` stellt ein *Singleton* dar, um doppelte Messdaten zu vermeiden.

Für den Empfang der Daten mittels *ANT* müssen bestimmte Parameter gesetzt werden. Diese Einstellungen sind variabel, weshalb sie in eine *XML* Datei ausgelagert sind, siehe Programm 6.1. Wenn in einem Raum mehrere intelligente Bürosessel genutzt werden, ist es notwendig die Sessel voneinander unterscheiden zu können, dies geschieht durch die *Channel Id* welche das *ANT* Protokoll zur Verfügung stellt. Die *Channel Id* setzt sich aus der *Device Number,* dem *Device Type* und dem *Transmission Type* zusammen, welche in der Konfigurationsdatei Programm 6.1 eingestellt werden. Die *Device Number* ist eine Nummer, über die man den Sender (hier Bürosessel) eindeutig identifizieren kann. Es können verschiedenste Arten von Geräten an einen Sender verbunden werden und die Daten der verschiedenen Geräte übermittelt werden. Damit man die empfangen Daten den einzelnen Geräten, zuordnen kann werden diese mit dem *Device Type* unterschieden. Mit dem *Transmission Type* werden Übertragungseigenschaften eines Geräte spezifiziert, welche am Sender (Bürosessel) eingestellt werden müssen. Damit der Empfänger eine Verbindung aufbauen kann, muss er diese Einstellungen

```xml
<?xml version="1.0" encoding="utf-8" ?>
<ant>
  <networknumber nr="0" />
  <channelId device="1" deviceType="42" transType="0" />
  <channel period="2184" />
  <radio frequency="66" />
</ant>
```

Programm 6.1: *XML* Konfigurationsdatei für *ANT*.

```
ANT_Device device = new ANT_Device();
antChannel = device.getChannel(0);
antChannel.assignChannel(
   ANT_ReferenceLibrary.ChannelType.BASE_Slave_Receive_0x00,
       networkNumber, 500);
antChannel.setChannelID(deviceNumber, pairingOn, deviceType,
transmissionType, 500);
antChannel.setChannelPeriod(msgPeriod, 500);
antChannel.setChannelFreq(radioFrequency, 500);
antChannel.setChannelSearchTimeout(searchTimeout, 500);
antChannel.channelResponse += new
   ANT_Channel.ChannelResponseHandler(
       this.ant_channel_channelResponse);
antChannel.openChannel();
```

Programm 6.2: Programmatisches setzen der Einstellungen des Empfängermoduls in C#.

kennen. Würde man die Sessel nicht voneinander unterscheiden können, würde dies zu einem Datenchaos führen, da es nicht möglich wäre die empfangenen Daten den einzelnen Sesseln zuzuordnen und somit wäre eine verlässliche Aussage über das Sitzverhalten unmöglich.

Durch die *Message Rate* (in der Konfigurationsdatei Programm 6.1 `period` unter `channel`) wird festgelegt, wie oft eine Nachricht verschickt wird. Die Nachrichten- oder Datenrate kann laut *ANT* Dokumentation[31] durch

$$\mathbf{Channel_Period_val} = \frac{\mathbf{32768}}{\mathbf{Message\ Rate(Hz)}}$$

ermittelt werden. Anhand der Konfigurationsdatei die in Programm 6.1 dargestellt ist, ergibt sich somit eine Datenrate von 15 Hz. Weiters muss die Frequenz eingestellt werden, mit der die Daten übertragen werden sollen. Die Standardeinstellung von *ANT* ist 66, was 2466 MHz entspricht. Diese Einstellung ist am Sessel voreingestellt und muss somit auch am Empfänger-Modul eingestellt werden. Die Einstellung erfolgt im `DataProvider` in dem die *XML* Datei mittels *Language Integrated Query* kurz *LINQ* eingelesen

[31] Genaue Dokumentation: http://www.thisisant.com/pages/developer-zone/ant-protocol-and-usage

wird. Die eingelesenen Einstellungen werden an eine erzeugte `ANT_Device` Klasse übergeben. Die notwendigen Klassen und Funktionen des *ANT* Protokolls werden über die *ANT DLL* bereitgestellt. In Programm 6.2 wird durch das Erzeugen einer neuen `ANT_Device` Klasse ein Empfängerstick am PC gesucht. Wird kein Stick gefunden, wird ein entsprechender Fehler geworfen. Das *ANT* System baut ebenfalls auf eine ereignisorientierte Programmierung auf, weshalb die Methode `channelResponse` zum Empfang der Daten implementiert werden muss. Programm 6.2 zeigt die Zuweisung einer lokalen Methode als `channelResponse` Methode. Durch den Aufruf von `openChannel()` wird der Datenaustausch zwischen dem Sendemodul am Bürosessel und dem Empfängermodul am PC hergestellt. Besteht eine Verbindung zwischen Sender und Empfänger wird bei jedem Empfang von einem Datenpaket die zugeordnete channel-Response Methode aufgerufen.

Der Sessel schickt im Abstand der Datenrate ein Datenpaket wie in Programm 6.3 dargestellt. Ein Datenpaket besteht aus acht Segmenten. Das erste Segment beinhaltet die eindeutige Sessel-Nummer. Das zweite und das dritte Segment beinhalten die *COPX* bzw. *COPY* Werte. Das vierte Segment enthält zurzeit keine Information. Im Falle eines Fehlers am *ACAM* wird in Segment fünf die entsprechende Fehlernummer (0 wenn kein Fehler aufgetreten ist) ausgewiesen. Segment sechs enthält die Kraft, welche auf die Sensoren ausgeübt wird (0 wenn keine Person auf dem Sessel sitzt). Segment sieben ist ein fortlaufender Zähler zwischen 0 und 255, der in zweier Schritten erhöht wird, wodurch Paketverluste identifiziert werden können. Das letzte Segment enthält Informationen über den Akkustatus am Sessel (fünf: voll aufgeladen, kleiner fünf: der Akku verliert Leistung, sechs: wird geladen). Die einzelnen Datensegmente werden auf gültige Werte überprüft. Das bedeutet, dass zuerst das fünfte Segment auf eine Fehlernummer überprüft wird und im Falle eines Fehlers wird dieser in einer Datei festgehalten und dieses Datenpaket verworfen. Ist kein Fehler aufgetreten, so wird die Akkuleistung überprüft. Ist die Leistung zu schwach, so ist eine reibungslose Datenübertragung nicht mehr gewährleistet. In diesem Fall wird eine entsprechende Meldung an die `MainLogic` weitergeleitet, welche diese Information an alle registrierten Programme/Klassen weitergibt. Wird der Akku aufgeladen, werden die Datenpakete lediglich in eine Datei geschrieben (um eine lückenlose Erfassung für die statistische Auswertung zu gewährleisten), aber nicht zur Auswertung der Sitzposition herangezogen, da eine gleichzeitige Messung beim Laden nicht möglich ist. Sobald der Akku geladen ist, wird wieder eine Meldung über die `MainLogic` an die registrierten Programme/Klassen geleitet, denn der Sessel muss über einen Neustart wieder in den Messmodus versetzt werden. Gibt die Akkuleistung keinen Grund zur Sorge, wird überprüft, ob Druck auf die einzelnen Sensoren ausgeübt wird (Segment sechs). Ist kein Druck auf den Sensoren, sitzt niemand auf dem Sessel, weshalb die

```
1 127 127 0 0 0 55 5
1 127 127 0 0 0 57 1
1 127 127 0 0 0 59 6
1 127 127 0 0 0 61 7
1 127 127 0 4 0 63 5
1 149 253 0 0 25 65 5
```

Programm 6.3: Empfangene Datenpakete vom intelligenten Bürosessel, jede Zeile entspricht einem Datenpaket und weist pro Segment verschiedene gültige Werte aus.

Messdaten lediglich in einer Datei abgelegt werden, um eine fortlaufende Aufzeichnung der Daten für die Studie zu gewährleisten. Außerdem werden nach einer vordefinierten Zeitspanne die Datenauswertungen und die aktuelle Sitzposition zurückgesetzt, solange sich niemand mehr auf den Sessel gesetzt hat. Wird eine sitzende Person anhand der Belastung erkannt, wird das Datenpaket über die `MainLogic` an den `DataManager` (mittels *Event*) weitergeleitet.

Bei Verbindungsproblemen gibt es die Möglichkeit, die Verbindung zurückzusetzten. Dies wird dadurch erreicht, indem die Parameter erneut eingestellt und der Kanal für den Empfang erneut durch den Aufruf `openChannel()` geöffnet wird. Bevor der Kanal wieder geöffnet wird, muss jedoch die Methode zum Empfangen der Daten (`channelResponse`) abgemeldet werden. Ansonsten besteht das Problem, dass bei einer erneuten Verbindung die Daten doppelt gespeichert werden, da die Methode zum Datenempfang in zwei verschiedenen `Threads` ausgeführt wird. Somit werden die Daten zwei Mal in einer Datei abgespeichert. Beim erneuten Öffnen des Kanals ist dieser im *Suchmodus*, um einen geeigneten Sender zu finden. Das Problem hierbei ist, dass das Empfangsmodul im *Suchmodus* bleibt, bis ein Sender gefunden ist. Deshalb wird ein `Timer` gestartet, welcher nach einer Minute ausgelöst wird, wenn bis dahin keine Daten empfangen wurden und löst ein *Ereignis* aus, dass die Verbindungsprobleme weiterhin bestehen.

DataManager

Wie die `MainLogic,` als auch der `DataProvider` ist der `DataManager` als *Singleton* realisiert, da bei mehreren Instanzen es möglich ist, dass Daten verloren gehen oder verdoppelt werden. Die Aufgabe des `DataManagers` besteht darin, die empfangenen Daten zu verwalten und auszuwerten. Bei der Auswertung der Daten muss besonders darauf geachtet werden, dass nur jene

```
public void AddData(Measurement m)

{
   lock (lst)
   {
      this.lst.Add(m);
   }
}
```

Programm 6.4: Kritischer Abschnitt mit *lock-Anweisung*. Die Sperre erfolgt auf dem Listenobjekt `lst`, in dem die Daten gespiechert werden, selbst.

Daten ausgewertet werden, die auch in die entsprechende Messperiode fallen, da das Sammeln der Daten parallel zur Auswertung erfolgt. Werden durch die `MainLogic` die Datenpakete an den `DataManager` geleitet, so werden diese Pakete zu einer Liste hinzugefügt. Beim Hinzufügen von Daten handelt es sich um einen so genannten kritischen Abschnitt, denn wenn neue Daten während der Auswertung hinzugefügt werden, würde dies die Auswertung verfälschen. Um eine Verfälschung zu verhindern, wird eine *lock-Anweisung*[32] (Programm 6.4) verwendet. Durch die Sperre auf das Listenobjekt `lst`, wird eine Abarbeitung dieses Programmsegmentes erst ausgeführt, wenn kein anderer Programmabschnitt oder *Thread* eine Sperre auf `lst` erhebt. Wird die Auswertung der Daten gestartet, so wird eine Sperre auf die Liste veranlasst. Danach werden alle Daten in dieser Liste an die Kostenfunktion übergeben und anschließend die Liste gelöscht (Programm 6.5). Dadurch ist gewährleistet, dass nur Daten innerhalb einer Messperiode ausgewertet werden. Nach dem Löschen der Daten aus der Liste wird die Sperre wieder aufgehoben und es können parallel zur Auswertung wieder Daten empfangen und gespeichert werden.

Das Ergebnis der Kostenfunktion ist die Basis für die nächsten *Ereignisse* in der Logik. Sitzt die Person schlecht, wird überprüft wie lange sie schlecht saß. Ist dieser Zeitraum noch im akzeptablen Bereich, wird dies an die `MainLogic` über ein *Event* weitergeleitet und somit auch an die registrierten Programme/Klassen. Wird die maximale Zeitspanne für eine andauernde schlechte Sitzhaltung überschritten, so wird eine Aufforderung zu einer Übungseinheit versendet. Wird eine gute bis sehr gute Sitzhaltung ermittelt, so wird dies ebenfalls mit einem *Ereignis* an die `MainLogic` und somit auch an die registrierten Programme und Klassen weitergeleitet. Der Zeitraum für

[32] http://msdn.microsoft.com/de-de/library/c5kehkcz(v=vs.8).aspx

```
public void AnalyseData()
{
...
   lock (lst)
   {
      foreach (Measurement m in this.lst)
      {
         costFunction.AddCop(m.GetCopX, m.GetCopY,
      m.GetWeight);
      }
      this.lst.Clear();
   }
...
}
```

Programm 6.5: Ausschnitt aus der Analyse Methode mit *lock-Anweisung*. Kopieren der Daten in die Kostenfunktion und anschließendes löschen.

schlechtes Sitzen wird aber nur um eine Messperiode verringert, um zu einem möglichst langanhaltenden, gutem Sitzverhalten anzuregen. Bei einer neutralen Sitzhaltung, also eine Zwischenstufe zwischen gut und schlecht, wird das Ergebnis der nächsten Messperiode abgewartet bevor *Ereignisse* versendet werden.

Wird die Trainingseinheit gestartet, so werden nur die Daten, die während der Übungseinheit gesammelt wurden, mit der Kostenfunktion ausgewertet. Das Ergebnis des Trainings wird mit den Schwellwerten für die jeweilige Übung verglichen. Ist die Übungseinheit sehr gut absolviert, liegt eine ausreichende Lockerung der Muskulatur und Entlastung der Wirbelsäule vor. Aufgrund dessen werden alle bisherigen Ergebnisse, sowie der gemessene Zeitraum für schlechtes Sitzen auf null zurückgesetzt. Ist das Ergebnis akzeptabel ausgefallen, so wird der Zeitraum auf die Hälfte der maximalen Zeitspanne für schlechtes Sitzen gesetzt. Bei einem schlechten Ergebnis der Übung, erfolgt keine Änderung der Zeitspanne. Das bedeutet, dass nach der nächsten Messperiode, erneut eine Trainingsaufforderung erfolgt, wenn sich die Sitzhaltung nicht verbessert hat. Ist das Ergebnis nach der nächsten Messperiode positiv, so wird wie vorhin beschrieben, die Zeitspanne für schlechtes Sitzen um eine Messperiode verringert. Dadurch soll eine aktive Sitzhaltung trainiert werden und bestenfalls unbewusst umgesetzt werden.

Bei Verbindungs- und Messproblemen wird auch der `DataManager` zurückgesetzt. Das bedeutet, dass die bisherigen Daten die an den

DataManager weitergeleitet wurden, verworfen werden. Dies wird gemacht, da Verbindungsprobleme des Öfteren auf einen Fehler im Datenpuffer beim Sende- oder Empfangsmodul zurückzuführen ist. Somit könnte es zu fehlerhaften Daten bei der Übermittlung gekommen sein. Um nicht mit falschen oder fehlerbehafteten Daten zu arbeiten, werden die Daten für die Messperiode, in der der Fehler auftrat, verworfen.

Event Klassen

Alle *Event*-Klassen implementieren die Schnittstelle IEvent (Abbildung 6.1) und dienen dem Informationsaustausch. Die *Event*-Klassen können in private und öffentliche unterteilt werden. Die privaten *Event*-Klassen werden für die Informationsweitergabe zwischen MainLogic, DataProvider und DataManager verwendet.

Die öffentlichen *Events* dienen dem Informationsfluss mit Programmen, die die Logik verwenden. Eines der wichtigsten Events stellt das BadPositionEvent dar, welches eine schlechte Sitzhaltung für die Messperiode angibt. Durch das PositionChangedEvent wird angezeigt, dass sich die Sitzhaltung gegenüber der letzten Messperiode verändert hat und somit ein dynamisches Sitzverhalten vorliegt. Mittels InitializeTrainingEvent wird zu einer Übungseinheit aufgefordert, um die statische Belastung über einen längeren Zeitraum auszugleichen. Wurde das Training absolviert, so wird ein TrainingRateEvent versendet, welches das Ergebnis der Übung beinhaltet. Das Ergebnis für die absolvierte Übung wird mit *Good*, *Ok* und *Bad* bewertet.

Während der Verwendung des Sessels, kann es zu Verbindungsproblemen kommen. Diese entstehen, wenn der Sessel zu weit vom Empfänger entfernt ist oder das *NEON Modul* während der Berechnung der *COPs hängt,* wodurch keine Datenübertragung mehr stattfindet. Die Sendeausfälle am Sessel kommen selten vor, dennoch müssen bei Verbindungsproblemen diese durch Abstandsverkleinerung zwischen Sessel und Empfänger oder durch Neustart des Sessels behoben werden, um weiterhin das Sitzverhalten bewerten zu können. Deshalb wird bei Verbindungsproblemen das ChairConnectionFailedEvent ausgelöst. Weiters gibt es noch Events, welche den Akkustatus angeben. Durch das InitializeLoadBatteryEvent wird signalisiert, dass der Akku fast leer ist und aufgeladen werden muss. Wird der Sessel mit dem Netzteil verbunden, wird das LoadingBatteryEvent gesendet, um das Laden des Akkus zu bestätigen. Ist der Akku aufgeladen, wird dies durch das FinishedLoadBatteryEvent mitgeteilt.

Durch das ExitEvent wird signalisiert, dass die Logik alle Berechnungen und die Verwaltung der Daten beendet hat. Das Programm, das die Logik verwendet, kann nun geschlossen werden.

Hilfsklassen

Die Hilfsklassen sind vor allem wichtig, um eine Verdoppelung von Programmcode und dadurch mögliche Fehlerquellen zu vermeiden. So wurde die Kostenfunktion, welche für die Berechnung der Sitzposition, als auch für die Berechnung des Trainingsergebnisses herangezogen wird, in eine eigene Klasse `Function` ausgelagert.

Die empfangenen Daten bilden die Grundlage für die Berechnungen und sind in einer Klasse `Measurement` zusammengefasst. Somit können die Daten leicht verwaltet und innerhalb der Logik weitergegeben werden, ohne jedes Mal das Datenpaket in Segmente zerlegen und in entsprechende Zahlenwerte umwandeln zu müssen.

Wie beschrieben, wurden die Einstellmöglichkeiten in zwei *XML* Dokumente ausgelagert. Verschiedenste Klassen in der Logik benötigen gewisse Einstellungen und müssen diese aus dem entsprechenden Dokument auslesen. Um Tippfehler in den Pfaden zum Dokument oder im Dokumentnamen zu verhindern wurde eine Klasse `ConfigurationUtils` angelegt. Diese Klasse bietet Methoden an, um für das gewünschte Konfigurationsdokument, den Pfad (inklusive Namen des Dokumentes) zu erhalten. Dies hat den Vorteil, dass der Pfad jederzeit in diesem Dokument geändert werden kann.

6.3 Benutzeroberfläche

Die hier beschriebene Benutzeroberfläche ist ein eigenständiges Programm, welches Teil des Prototypen für die Studie ist. Es basiert wie die Logik auf dem *.Net Framework 4*. Die grafischen Elemente wurden mit der *Windows Presentation Foundation*[33] (*WPF*) in *C#* realisiert.

6.3.1 Grundlegende Funktionsweise

Nach dem sich das Programm bei der Logik für die Informationsweitergabe (*Ereignisse/Events)* registriert hat, wird nach dem Start, je nach Art der Rückmeldung, das grüne Symbol für die Sitzhaltung oder ein neutrales Icon gesetzt. Handelt es sich bei den *Ereignissen* um jene, die die Sitzhaltung betreffen, so wird die Farbe des Icons oder die Neigung der *Hanappa*, je nach Sitzhaltung, verändert. Wird durch ein *Event* zu einer Übungseinheit aufgefordert, wird dies mittels einer Animation an den Benutzer bei der graphischen Rückmeldung weitergegeben – siehe Abbildung 3.9 – bzw. wird bei der physischen Rückmeldung ein Fenster minimiert geöffnet, welches zusätzlich

[33] http://msdn.microsoft.com/en-us/library/ms754130.aspx

blinkt und die *Hanappa* bewegt sich. Bei Verbindungsproblemen mit dem Sessel oder der *Hanappa*, sowie bei Veränderungen des Akkustatus werden auch hier mittels Animationen diese Informationen für den Benutzer angezeigt.

6.3.2 Allgemeines

Durch die Verwendung von *WPF* muss für ein lauffähiges Programm, gewisse Grundsätze beachtet werden. So wird das Aussehen durch die *Extensible Application Markup Language* kurz *XAML*, beschrieben. Mit *Visual Studio* bzw. *Expression Blend 4* [34] ist es durch die integrierten *UI Builder* möglich, die Bedienelemente einfach zu gestalten. *Visual Studio* und *Expresion Blend 4* genieren den nötigen *XAML* Code für die Anwendung. Dennoch ist in manchen Fällen eine händische Nachbearbeitung nötig, um das gewünschte Ergebnis zu erzielen – siehe Animationen 4.3.4. Der Programmcode wird in einer separaten Datei gespeichert und ist somit von der *UI* Beschreibungssprache abgekoppelt. Ein Vorteil hierbei ist, dass beim Einsatz von *Visual Studio* in Kombination mit *Expression Blend* parallel mit beiden Programmen an einer Anwendung gearbeitet werden kann. Um eine gültige Anwendung zu erzeugen, muss ein Fenster benannt werden, welches beim Start der Anwendung erzeugt wird. Wird dieses Hauptfenster geschlossen, wird das gesamte Programm beendet.

6.3.3 Komponenten

Hauptfenster

Das Fenster `MainWindow` ist das Hauptfenster und wird beim Start der Anwendung erzeugt. Dieses Fenster dient wie die `MainLogic` Klasse der Logik als zentrale Verwaltung der Anwendung. Es übernimmt die Steuerung der anderen Fenster und macht für diese die nötigen Informationen zugänglich oder leitet sie weiter. Das Fenster selbst beinhaltet keine Steuerelemente und wird nach dem Starten der Anwendung auf *unsichtbar* geschaltet, wodurch es am Bildschirm nicht angezeigt wird. Das Hauptfenster instanziert die Logik über die bereitgestellte `LogicFactory` (*Factory Method*). Im Programm wird nur mit der Schnittstelle und den darin enthaltenen Methoden gearbeitet, wie in Programm 6.6 ersichtlich. Nach dem nun eine Instanz der Logik in der Klasse vorliegt, kann sich das `MainWindow` für *Ereignisse* bei der Logik

[34] Grafik Programm zum Gestalten von *UIs* und Animation für WPF Anwendungen. http://www.microsoft.com/germany/expression/products/Blend-Overview.aspx

```
1 ILogic logic = LogicFactory.GetLogic();
2 logic.AddListener(this);
```

Programm 6.6: In Zeile eins wird die Instanzierung der Logik durch die Factory Methode durchgeführt. Die zweite Zeile zeigt das Registirieren für *Ereignisse* der Logik im `MainWindow` (=this).

registrieren. Um die *Ereignisse* auch erhalten zu können, wird die Methode `HandleEvent(IEvent e)` implementiert.

Erhält das Hauptfenster *Ereignisse* von der Logik, wird überprüft welche Art von *Ereignis* empfangen wurde. Handelt es sich bei dem *Event* um allgemeine Informationen über den Status des Sessels, so behandelt das Hauptfenster dieses *Ereignis*. So wird im Falle eines schwachen Ladezustandes des Akkus, das entsprechende Fenster sichtbar geschaltet und durch eine Animation wird das Fenster seitlich in den Bildschirmbereich bewegt. Ebenso wird bei einem Verbindungsfehler ein entsprechendes Fenster auf sichtbar gestellt und die entsprechende Animation abgespielt. Das Hauptfenster ist auch für das *Ereignis*, welches zur Aufforderung einer Übung gesendet wird, zuständig. Wie in Abbildung 5.1 dargestellt, wird das Fenster `Training` angezeigt und animiert, sofern die graphische Rückmeldung aktiviert ist. Andernfalls wird wie in Kapitel 4 beschrieben, ein blinkendes Fenster minimiert geöffnet und die *Hanappa* bewegt sich.

Enthält das *Ereignis* Informationen über die Sitzhaltung, so wird dieses an den Avatar – hier das Rückmeldesystem - weitergeleitet (Programm 6.7). Der Hintergrund besteht darin, dass je nach Einstellung ein Avatar, die *Hanappa* oder die graphische Rückmeldung verwendet wird und unterschiedliche Anweisungen beinhalten – siehe Avatarklassen. Wichtig hierbei ist zu beachten, dass wenn keine Rückmeldung eingestellt ist, auch keine Avatarklasse angelegt wird. Daher muss bevor das *Ereignis* weitergeleitet wird, überprüft werden, ob eine Avatarklasse vorhanden ist oder nicht.

Das `MainWindow` verwaltet Informationen, welche von mehreren Klassen benötigt werden. Da es mehrere Rückmeldesysteme gibt, wurde der aktuelle Status über das Sitzverhalten im Hauptfenster gespeichert. Durch die Zentralisierung reagiert jedes Rückmeldesystem auf die *Ereignisse* gleich, wodurch eine Verdoppelung des Programmcodes und somit eine potenzielle Fehlerquelle vermieden wurde.

Für die vier verschiedenen Übungseinheiten gibt es Videos, damit der Benutzer weiß, welche Bewegung er ausführen soll. Pro Übung sind vier Videodateien vorhanden. Auf den Videos ist die gleiche Übung zu sehen, jedoch ist die Person, die die Übung im Video ausführt eine andere. Damit

```
1  public void HandleEvent(IEvent e)
2  {
3     if (e is InitializeLoadBatteryEvent)
4     {
5        ...
6     }
7     ...
8     else
9     {
10       if (this.avatar != null)
11          this.avatar.HandleEvent(e);
12    }
13 }
```

Programm 6.7: Abarbeitung der *Ereignisse* in der `MainWindow` Klasse. Die Zeilen 10 und 11 zeigen, die Weitergabe des *Events* an die `Avatarklasse`.

das Training nicht eintönig wird und der Benutzer die Abfolge der Übungen erlernen kann, wird über ein Zufallsprinzip die nächste Übung ermittelt, wie in Programm 6.8 dargestellt. Dies wird erreicht, indem alle Dateinamen der Videos (insgesamt 12) in einer Liste gespeichert werden. Der Dateiname setzt sich aus einem Namen (für die Person im Video) und einer Nummer (für die Übungsaufgabe) zusammen. Mit Hilfe der *.Net* Klasse `Random`, wird aus dieser Liste ein Dateiname ausgewählt. Um zu verhindern, dass die gleiche Übung ein- oder mehrmals hintereinander ausgeführt werden soll, wird der Name, des zuletzt abgespielten Videos gespeichert. Der gespeicherte Dateiname wird mit dem neu ermittelten Zufallsnamen verglichen. Gibt es

```
internal void ChangeVideoSelection()
{
   Random random = new Random();
   this.selection =
videoSelection[random.Next(this.videoSelection.Count-1)];

...
}
```

Programm 6.8: Auswahl eines Dateinamens durch Zufallsgenerierung mittels `random.Next`.

eine Übereinstimmung wird eine andere Übung ausgewählt. Weiters wird auch darauf geachtet, dass im Video nicht zweimal hintereinander die gleiche Person zu sehen ist, die die Übung vormacht (siehe Anhang A Programm A.2 und Programm A.3).

Treten Fehler seitens der Verbindung oder Messung beim Bürosessel auf, wird das Programm über die Logik mit einem *Ereignis* darauf aufmerksam gemacht. Eine weitere Fehlerquelle ist die Verbindung zur *Hanappa* über *USB*. Ist der das *USB*-Kabel nicht richtig mit der *Hanappa* verbunden oder ausgesteckt, so wird eine `Exception` geworfen. In jedem dieser Fälle wird ein entsprechendes Fenster angezeigt. Die Möglichkeit, dass diese Fehler gleichzeitig auftreten ist nach dem Starten des Programmes am Wahrscheinlichsten, da der Sessel nicht eingeschaltet und das *USB*-Kabel der *Hanappa* beim Starten des PC's versehentlich entfernt worden sein könnte. Es muss daher darauf geachtet werden, dass sich die Fehlermeldungen nicht überlagern. Deshalb wird, bevor eine Fehlermeldung angezeigt wird, überprüft, ob ein anderes Fenster bereits sichtbar ist. Ist dies der Fall, so wird ein `Timer` gestartet, der nach einer Minute erneut prüft, ob das Fenster noch angezeigt wird. Ist dies der Fall so wird der `Timer` erneut nach einer Minute wieder aufgerufen. Sind keine anderen Fenster sichtbar, wird die ausstehende Fehlermeldung angezeigt. Eine festgelegte Reihenfolge, welcher Fehler zuerst angezeigt wird, gibt es nicht. Hier wird nach dem *First come, first serve* Prinzip gehandelt, das bedeutet, dass jener Fehler der zuerst erkannt wird, auch zuerst angezeigt wird.

Ein wichtiger Punkt für die Studie stellt den Störeinfluss der Übung auf den Arbeitsfluss dar. Bei der Implementierung wurde darauf geachtet, dem Benutzer eine Wiederaufnahme seiner Tätigkeit so leicht wie möglich zu machen (transparenter Hintergrund, Selbstbestimmung über den Zeitpunkt wann die Trainingseinheit absolviert wird). Um der Wiederaufnahme eine greifbare Größe zu geben, wurde ein sogenannter *Hook* [35] gesetzt. Über diesen *Hook* können verschiedene Systemnachrichten mitgehört werden. Für die Studie wurde eine Wiederaufnahme der Tätigkeit vor dem Training wie folgt definiert:

- Klick in ein Fenster, welches nicht zum Prototypen gehört und
- Tastatureingabe in einem Fenster, welches nicht dem Prototypen zugeschrieben werden kann.

Wird nach dem Training eine Systemnachricht empfangen, die auf einen der beiden Punkte zutrifft, wird die Zeitspanne berechnet, wie lange es seit dem Beenden des Trainings bis zum Auftreten des *Hooks* gedauert hat. Diese

[35] http://msdn.microsoft.com/en-us/library/ms632589(v=VS.85).aspx

Zeitdifferenz wird in einer entsprechenden Datei abgelegt – siehe Logging.

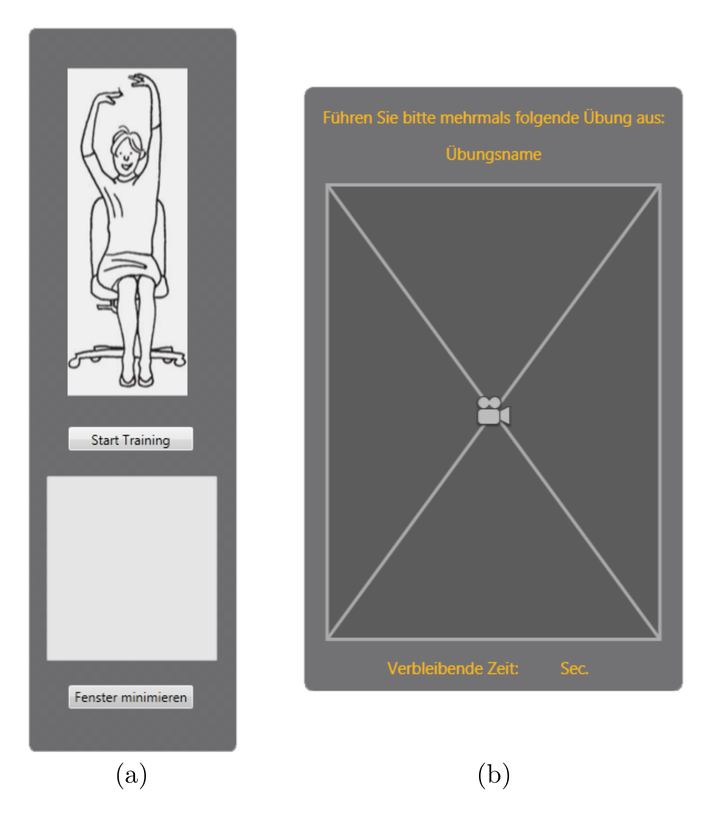
Wird das Programm beendet, werden die zugehörigen Fenster (bis auf das `MainWindow`) geschlossen. Anschließend wird auch die Logik vom Beenden des Programmes in Kenntnis gesetzt, durch Aufruf der `ExitLogic()` Funktion. Das Beenden der Logik wird abgewartet, um keinen Datenverlust zu erleiden. Hat die Logik das Beenden mittels `ExitEvent` bestätigt, werden die benötigten Ressourcen (u.a. der *Hook*) freigegeben und das Logging beendet. Nach diesen *Aufräumarbeiten* wird auch das `MainWindow` geschlossen.

Trainingsfenster

Das Trainingsfenster besteht, wie Abbildung 6.3 zeigt, aus zwei Elementen Ein Element wird für die Anzeige vor und nach der Übung verwendet, während das zweite für das Abspielen des Videos und der Anzeige der Dauer der Übung benötigt wird.

Das erste Element beinhaltet beim Erzeugen ein leeres Textfeld, welches mit einem entsprechenden Text versehen wird, je nachdem ob es zum Training auffordert oder das Training beendet ist. Wird zur Übung aufgefordert, so wird gleichzeitig ein `Timer` aktiviert. Dieser `Timer` läuft nach knapp einer Minute ab. Wurde die Übung nicht gestartet, so wird das Fenster automatisch wieder durch eine Animation ausgeblendet. Dies dient dazu, dass der Benutzer darauf aufmerksam gemacht wird, dass sein Sitzverhalten in den letzten 21 Minuten schlecht war und eine Übung zur Verbesserung durchgeführt werden sollte. Bearbeitet der Benutzer eine wichtige Aufgabe, bietet das Programm so die Möglichkeit, dass der Benutzer weiterarbeiten kann ohne das Fenster selbst schließen zu müssen. Der Benutzer kann durch diese Aufforderung ein besseres Sitzverhalten anstreben. Ändert sich das Sitzverhalten in der nächsten Messperiode nicht, so wird erneut zu einer Übungseinheit aufgefordert. Verbessert sich das Sitzverhalten, so wird erst wieder eine Aufforderung angezeigt, wenn insgesamt sieben Messperioden lang ein schlechtes Sitzverhalten vorliegt.

Wird das Training gestartet, so übernimmt das zweite Element die Anzeige. Durch eine Übergangsanimation werden die Elemente ausgetauscht. Während diesem Übergang werden Vorkehrungen für das Abspielen des Videos sowie für den Countdown für die Trainingsdauer getroffen. Ist dieser Übergang vorbei so wird das zuvor geladene Video abgespielt und der Zähler für die Übungsdauer wird im Sekundentakt verringert. Ist der Zähler bei null angelangt, so werden durch eine Animation die Anzeigeelemente wieder vertauscht. Anstatt einer Trainingsaufforderung wird der Text auf das absolvierte Training hin geändert. Außerdem wird die Schaltfläche zum Starten des Trainings deaktiviert.



(a) (b)

Abbildung 6.3: Elemente des Trainingsfensters ohne programmatische Einstellung. (a) Die Trainingsaufforderung ohne den definierten Text. (b) Element zum Abspielen des Videos, mit einem Platzhalter für den Namen der Übung, für das Video, sowie für einen Zähler, der die Restdauer des Trainings angibt.

Informationsfenster

Werden *Ereignisse* bezüglich des Akkuladezustandes empfangen, so wird das entsprechende Informationsfenster mit einer Animation angezeigt. Ist der Akkuladezustand zu niedrig, wird zum Aufladen aufgefordert. Wird erkannt, dass der Akku geladen wird, wird dies auch angezeigt um dem Benutzer eine Rückmeldung nach dem Einstecken zu geben. Ist der Akku vollständig aufgeladen, so wird dazu aufgefordert, den Sessel wieder vom Ladegerät zu trennen und einzuschalten.

Gibt es jedoch einen Verbindungsfehler mi der *Hanappa*, so wird ein entsprechendes Fenster angezeigt, wie in Abbildung 6.4 (b) dargestellt. In diesem Fenster wird der Benutzer aufgefordert, die Steckverbindungen zu prüfen. Dieses Fenster wird nur dann angezeigt, wenn als Rückmeldung die *Hanappa* in der Konfigurationsdatei eingestellt ist.

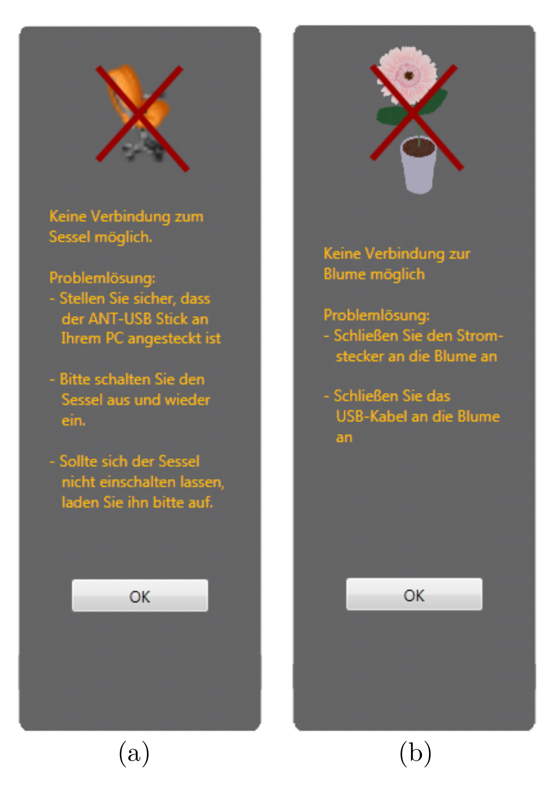


Abbildung 6.4: Fehlermeldungen mit Anweisung zur Behebung des Fehlers. (a) Meldung bei Verbindungsproblemen mit dem Sessel. (b) Meldung bei Problemen mit der *Hanappa*.

Liegt ein Verbindungsproblem mit dem Sessel vor, wird auch in diesem Fall eine entsprechende Meldung – Abbildung 6.4 (a) – mit Anweisung zum Beheben des Fehlers angezeigt. Dieses Fenster wird nicht automatisch nach einer gewissen Zeit ausgeblendet, da der Klick auf die OK-Schaltfläche einen Neustart der Logik auslöst, wodurch das Verbindungsproblem behoben wird.

Avatarfenster

Das Avatarfenster wird benötigt, wenn die graphische Rückmeldung eingestellt ist. Dieses Fenster beinhaltet das entsprechende Icon je nach Sitzhaltung. Des Weiteren stellt das Fenster drei Methoden zur Verfügung, um das Icon zu verändern. Die Methode `ChangeTaskIconUp()` wird verwendet, wenn sich die Sitzhaltung verbessert hat. Dadurch wird die Farbe schrittweise bis hin zur grünen Farbe geändert, um die aktuelle Sitzhaltung widerzuspiegeln. Mittels `ChangeTaskIconDown()` wird genau der gegenteilige Effekt erzielt und ist repräsentativ für eine Verschlechterung des Sitzverhaltens. Mit der

dritten Methode, `ChaneIcon(Uri iconUri)`, kann das Icon explizit angegeben werden. Dadurch kann das Icon je nach Trainingsergebnis gesetzt werden und stellt damit ein Feedback für den Benutzer dar, wie gut er die Übung absolviert hat.

Avatarklassen

Die Avatarklassen, `VirtualHanappa` und `PhysicalHanappa`, repräsentieren das jeweilige Rückmeldesysteme. Sie verwalten die Programmlogik, wie mit den *Ereignissen* zur Sitzhaltung umgegangen werden soll. Grundsätzlich wäre es möglich, diesen Teil auch im `MainWindow` zu implementieren, wo die *Ereignisse* empfangen werden. Der Grund diesen Teil in eigene Klassen auszulagern besteht darin, dass nicht erst überprüft werden muss, welche Rückmeldung eingestellt ist, sondern es können *Ereignis* einfach an die Avatarinstanz weitergeleitet werden. Dies verringert die Komplexität des Programmcodes und gleichzeitig wird die Lesbarkeit des Codes erhöht. Deshalb wurde das Erzeugen des Avatars als *Factory Method* implementiert. Somit ist im `MainWindow` nur eine Instanz des Avatars vorhanden, welche der Rückmeldung entspricht und man kann das *Ereignis* an diese Instanz weitergeben. Deshalb implementiert jede Avatarklasse die Schnittstelle `IAvatar`, wodurch die Funktionalität klar vorgegeben ist. Über die `AvatarFactory` kann man die Instanz für den Avatar anfordern, wodurch sichergestellt ist, dass die eingestellte Rückmeldung aktiviert ist.

Die `VirtualHanappa` regelt die Darstellung des Icons im `Avatarfenster` entsprechend des Sitzverhaltens oder des Übungsergebnisses. Die `PhysicalHanappa` steuert die *Hanappa*-Blume. Die Steuerung erfolgt über den eingestellten `SerialPort`. So werden bei einer schlechten Sitzhaltung pro *Event null*, *eins* und *zwei* auf den `SerialPort` geleitet. Dadurch neigen sich die Blüte und die zwei Blätter immer weiter nach unten, wodurch der Eindruck entsteht, die *Hanappa* würde verblühen. Durch Weitergabe von a, b und c an den `SerialPort` richtet sich die *Hanappa* schrittweise wieder auf. Bei einer Trainingsaufforderung werden diese Befehle abwechselnd gesendet, wodurch sich die Blüte und die Blätter auf und ab bewegen und es aussieht, als ob die *Hanappa* „Gymnastik“ macht.

Logging

Es werden alle *Ereignisse* die von der Logik empfangen werden, mit Hilfe der `EventLogger` Klasse gespeichert. Diese Klasse ist als *Singleton* konzipiert und kann von jeder Klasse innerhalb des Programmes verwendet werden. Neben dem Erfassen des Zeitpunktes wann das *Ereignis* empfangen wurde und dem

Namen des *Ereignisses* werden für die statistische Auswertung auch bestimmte Faktoren festgehalten. So werden neben den *Ereignissen* der Logik auch vom Benutzer ausgelöste *Ereignisse* festgehalten, beispielsweise wann die Übung gestartet wurde. Weiters wird festgehalten, wie lange es von der Trainingsaufforderung dauerte, bis der Benutzer die Übung absolvierte.

6.3.4 Animationen

Animationen in *WPF* werden über sogenannte `Storyboards` beschrieben. Sie können über *XAML* definiert oder mit *C#* programmatisch erstellt werden. Da selbst einfache Animationen schon eine beachtliche Anzahl an Code-Zeilen (sei es *XAML* oder *C#*) bedeutet, wurde für den Prototypen auf *Expression Blend 4* zurückgegriffen. Dies hat den Vorteil, dass man über eine grafische Oberfläche schnell Animationen gestalten kann und der dazugehörige *XAML* Code wird automatisch generiert. Somit wurden sämtliche Animationen des Programmes innerhalb von kurzer Zeit erstellt.

Die automatische Erzeugung des Codes ist nicht perfekt, wodurch auch selbst noch in den Code eingegriffen werden muss. So wird zum Beispiel automatisch ein *Trigger* erzeugt, welcher festlegt, wann die Animation gestartet werden soll, wie Programm 6.9 zeigt. Diese erzeugten *Trigger* wurden gelöscht, um bei entsprechenden *Ereignissen* von der Logik, die Animationen jeder Zeit aus dem Programmcode starten zu können.

Weiters musste für das Expandieren des Hintergrundes für das Trainingsvideo auch in die Animation eingegriffen werden. Mit *Expression Blend 4* kann für den Hintergrund eine fixe Breite für den Zeitpunkt zu Beginn der Animation und für das Ende eingestellt werden. Dadurch verbreitert sich der Hintergrund beim Abspielen der Animation. Das Problem hierbei ist, dass der Bildschirm des Benutzers nicht immer dieselbe Größe aufweist. Deshalb muss die Breite während der Laufzeit des Programmes ausgelesen und eingestellt werden, Programm 6.10 erläutert wie dies erreicht wird. Im Programmcode wird eine eigene Animation `myAnimation` angelegt und die entsprechenden Parameter `myAnimation.From` und `myAnimation.To` eingestellt. Dadurch wird

```
<Window.Triggers>
    <EventTrigger RoutedEvent="FrameworkElement.Loaded">
        <BeginStoryboard Storyboard="{StaticResource move}"/>
</EventTrigger>
</Window.Triggers>
```

Programm 6.9: Beispiel für einen von *Expression Blend 4* erzeugten *Trigger*, der festlegt, wann die Animation starten soll.

```
DoubleAnimation myAnimation = new
DoubleAnimation();myAnimation.From = this.rectangle.Width;
myAnimation.To = Screen.PrimaryScreen.WorkingArea.Width;
myAnimation.BeginTime = TimeSpan.FromSeconds(0);
myAnimation.Duration = new
    Duration(TimeSpan.FromSeconds(2));
Storyboard.SetTargetName(myAnimation, this.rectangle.Name);
Storyboard.SetTargetProperty(myAnimation, new
PropertyPath(Rectangle.WidthProperty));
Storyboard sb = (Storyboard)this.Resources["StartVideo"];
sb.Children.Add(myAnimation);
```

Programm 6.10: Erzeugung einer neuen Animation myAnimation für die Verbreiterung des Hintergrundes und Hinzufügen zur ursprünglichen Animation StartVideo.

festgelegt von welchem Punkt aus der Hintergrund verbreitert werden soll und wo der Endpunkt liegen soll. Diese Animation muss nun zur ursprünglichen Animation `StartVideo` hinzugefügt werden, damit sie gemeinsam ausgeführt werden. So wird das `Storyboard` der `StartVideo` Animation über die `Resources` geladen und die neu erstellte Animation als Kindelement angefügt.

6.4 Datenspeicherung

Wie in den vorherigen Kapiteln erwähnt, werden gewisse Daten zur statistischen Auswertung gespeichert. So wird in den *Eigenen Dateien* am Rechner des Benutzers ein Ordner *Logging*, wie in Abbildung 6.5 dargestellt, erzeugt. Innerhalb dieses Ordners befindet sich ein Unterordner *userX*. Das *X* repräsentiert die in der Konfigurationsdatei festgelegte Nummer, die jedem Probanden zu Beginn der Studie zugeteilt wurde. Eine Ebene tiefer werden Ordner für die jeweils zu speichernden Daten angelegt:

- *COPs.* Speicherung der Rohdaten die vom Sessel empfangen wurden.
- *CalcResults.* Die Ergebnisse der Kostenrechnung
- *Events.* Die *Ereignisse*, welche von der Logik empfangen bzw. durch den Benutzer ausgelöst wurden.
- *Error.* Die Nummer des Fehlers am *ACAM*.
- *Info.* Es wird festgehalten, wann welche Rückmeldung eingestellt ist und das Geschlecht des Probanden.
- *Training.* Die Ergebnisse der absolvierten Übung

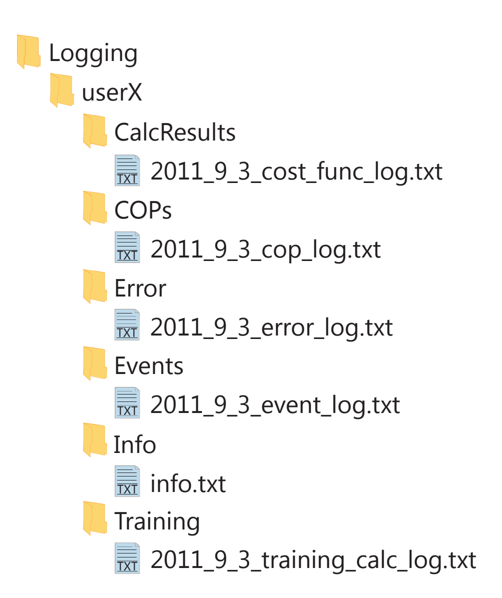


Abbildung 6.5: Dateistruktur.

Alle in diesen Ordner befindlichen Dateien, mit Ausnahme der *info.txt* weisen folgende Namenskonvention:

Jahr_Monat_Tag_Name_log.txt

Für jeden Tag wird ein eigenes Textdokument mit dieser Konvention angelegt. Dadurch sind die Dateien schon vorsortiert.

Lediglich die *info.txt* wird nur einmal angelegt und die Informationen pro Tag einmal hineingeschrieben, um die allgemeinen Informationen beim Öffnen auf einen Blick zu sehen.

Kapitel 7

Diskussion

In diesem Kapitel werden Probleme, die während der Ausarbeitung des Projektes auftraten, erläutert. Weiters werden Lösungen bzw. Lösungsansätze präsentiert um die genannten Probleme zu beheben. Da es sich um einen Prototyp handelt, wurden manche Probleme provisorisch gelöst und müssen für den Dauereinsatz als Produkt noch überarbeitet werden.

7.1 Intelligenter Bürosessel

Während der Testphasen hat sich gezeigt, dass der Sessel von Zeit zu Zeit keine korrekten Daten mehr geliefert hat. Eine erneute Kalibrierung hat dieses Problem behoben. Es stellte sich heraus, dass durch die Reibung der Kleidung auf dem Sesselbezug elektrische Spannung entsteht und sich diese nach einiger Zeit entlädt. Dies hat einen negativen Einfluss auf die angebrachte Elektronik am Sessel. Deshalb wurden die Bauteile in einem metallischen Gehäuse verbaut als Abschirmung gegen die elektrische Entladung. Dies wiederum beeinträchtigte die Sendeleistung. Deshalb musste in manchen Fällen der *USB*-Empfänger mit einer Verlängerung an einer geeigneten Stelle angebracht werden, um eine reibungslose Datenübertragung sicherzustellen.

Weiters gab es Probleme mit der Anzeige des Akkuladezustandes während des Aufladens. Beim Laden des Akkus sollte das Datensegment eine 6 beinhalten. Wenn der Akku aufgeladen ist, sollte eine 7 dies ersichtlich machen. Das Bauteil, das für die Messung des Zustandes des Akkus zuständig ist, hat jedoch in dieser Fabrikation einen Fehler, wodurch eine genaue Messung während des Ladens nicht möglich war. So schwankte die Anzeige des Ladzustandes während des Ladevorgangs zwischen 6 und 7. Dies machte eine Aussage über den Ladestatus unmöglich. Deshalb wurde getestet wie lange der Akku bis zur vollständig Ladung benötigt. Nach 2,5 bis 3 Stunden hatte der Akku seine volle Leistung wieder erreicht. Dieser Fehler wurde in der Software abgefangen, indem ein `Timer` gestartet wurde. Nach dem dieser ab-

gelaufen war, wurde das *Event*, welches signalisiert, dass der Akku geladen ist, ausgelöst.

Während der Testphase wurde festgestellt, dass bei manchen Sesseln die Akkuleistung schneller nachließ. Ein genauer Grund konnte nicht eruiert werden, jedoch wurde dieser Umstand als nicht kritisch für die Studie eingestuft.

7.2 Software

Das Empfangen der Daten wird in einem eigenen *Thread* ausgeführt. Liegt ein Verbindungsproblem mit dem Sessel vor, so kann es vorkommen, dass ohne laufenden Datenstrom, der *Thread* beendet und die Ressourcen freigegeben werden. Sendet der Sessel nach einem Neustart wieder Daten, so können diese nicht empfangen werden, da der entsprechende *Thread* nicht mehr existiert. Deshalb wird nach jedem empfangenen Datenpaket ein `Timer` gestartet, welcher im Abstand von drei Sekunden aktiviert wird. Ist nach drei Sekunden kein Datenpaket empfangen worden, so wird überprüft wie lange keine Daten empfangen wurden. Überschreitet die Zeitspanne eine Minute wird ein *Ereignis* ausgelöst, um auf den Verbindungsfehler aufmerksam zu machen. Der Grund für das verspätete Anzeigen der Fehlermeldung ist, dass der Benutzer kurz mit dem Sessel an eine andere Position gefahren ist, um Unterlagen oder ähnliches zu holen. Der `Timer` bleibt solange aktiv, bis ein Datenpaket empfangen wurde, um den *Thread* am Laufen zu halten.

In der Testphase hat sich herausgestellt, dass das Abspielen der Übungsvideos zu Problemen geführt hat. So wurde das Video auf manchen Rechner nicht abgespielt oder es wurde nicht flüssig wiedergegeben, gleichzeitig stieg die Prozessorauslastung stark an. Da kein Einfluss auf die Hardware der Geräte der Probanden genommen werden kann, wurden die Videos verkleinert und komprimiert. Dadurch wurde an Qualität eingebüßt, jedoch konnten die Videos auch auf leistungsschwachen PC's wiedergegeben werden.

Ein weiteres Problem mit dem Abspielen der Videos wurde ausgelöst, wenn der *ffdshow Codec* [36] auf dem Computer installiert ist. In Abbildung 7.1 wird die Meldung gezeigt, die beim Abspielen des Trainingsvideos angezeigt wird, welche verschiedene Optionen zur Auswahl stellt. Bei einer falschen Auswahl wird das Video nicht abgespielt oder im schlimmsten Fall führt dies zu einem Absturz von Windows. Um den korrekten Ablauf zu gewährleisten, wurde ein Benutzerhandbuch erstellt. Neben allgemeinen Funktionsweisen des Programmes und der Rückmeldungen, enthielt sie die korrekte Einstellung für diese Meldung. Im Falle einer Weiterentwicklung sollte eine Lösung gefunden werden, die das Anzeigen der *ffdshow* Meldung verhindert.

[36] http://ffdshow-tryout.sourceforge.net/

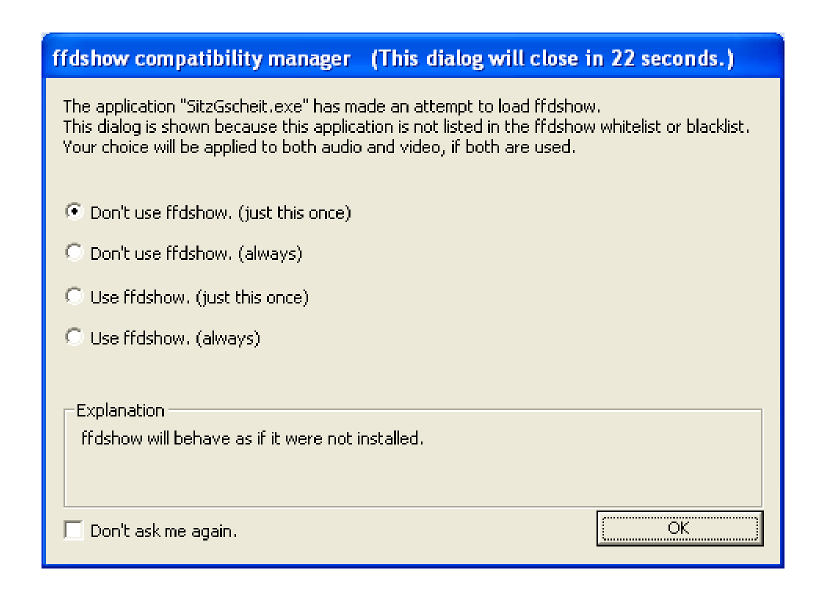


Abbildung 7.1: Meldung beim Abspielen des Videos, wenn *ffdshow* installiert ist, mit der korrekten Auswahl der Einstellung.

Beim Anzeigen des entsprechenden Informationsfensters über den Akkuzustand trat ebenfalls ein Fehler auf. Die unterschiedlichen Informationen befanden sich zuerst in einem Fenster. Für jede Information in diesem Fenster wurde mit *Expression Blend 4* eine eigene Animation erstellt. Damit im Programmcode die Animationen richtig geladen und abgespielt werden können, müssen diese über einen eindeutigen Namen identifiziert werden. In *Expression Blend* wurden die Animationen über den eindeutigen Namen korrekt abgespielt. Beim Abspielen in der Anwendung wurde immer die gleiche Animation abgespielt, gleich welche Animation angegeben wurde. Eine Ursache für dieses Verhalten konnte nicht gefunden werden, da es auch keine Fehlermeldung gab. Dieses Problem wurde behoben, indem die einzelnen Anzeigeelemente jeweils in ein eigenes Fenster ausgelagert wurden, mit einer entsprechenden Funktion zum Abspielen der Animation.

Beim ersten Test des Programmes kam es bei der Bearbeitung der Logik-*Ereignisse* seitens der Benutzeroberfläche zu einem Absturz. In *WPF* Anwendungen kann nur der sogenannte *UI Thread* die erzeugten Objekte der Benutzeroberfläche verändern. Die *Ereignisse* werden von der Logik in einem von ihr erzeugten *Thread* versandt, wodurch zum Beispiel das Ändern des Icons zur Anzeige des Sitzverhaltens nicht möglich ist, da es sich dabei nicht um den *UI Thread* handelt. Deshalb muss der *Ereignis-Thread* mit dem *UI Thread* synchronisiert werden, wie in Programm 7.1 dargestellt. Durch die

```
mw.Dispatcher.Invoke(System.Windows.Threading.DispatcherPrio
rity.Normal, new Action(
   delegate()
   {
      mw.ChangeIcon(null);
   }
));
```

Programm 7.1: Synchronisierung mit dem *UI Thread*, um das Icon zu verändern.

Verwendung von `Dispatcher.Invoke(Priority, Delegate Method)` [37] wird der entsprechende Programmteil mit dem *UI Thread* synchronisiert und das Icon kann verändert werden.

Ein weiteres Problem stellten die Benutzerrechte am Computer der Probanden dar. Während das Problem des Speicherns der notwendigen Daten durch Änderung des Speicherortes gelöst wurde, gab es Probleme beim Herunterfahren von Windows. Da nicht vorgesehen ist, den Prototypen durch das Klicken auf die Beenden-Schaltfläche zu stoppen, wird das Programm beim Herunterfahren von Windows beendet. Da es bei der *Hanappa*, bei zu langem Gebrauch bzw. Stromzufuhr, zu Überhitzungsproblemen kommen und der Akku im besten Fall zwei Tage ohne Aufladen den Sessel mit Strom versorgen kann, sollte der Benutzer erinnert werden, den Akku aufzuladen und die Stromzufuhr der *Hanappa* zu unterbrechen. Dies sollte durch ein Erinnerungsfenster erfolgen, nach dem das Herunterfahren von Windows abgebrochen wurde. Durch Klick auf die Okay-Schaltfläche im Erinnerungsfenster sollte die Anwendung beendet und das Herunterfahren automatisch erfolgen. Durch die Benutzerrechte, gab es jedoch eine Fehlermeldung beim Stoppen des Herunterfahrens. Obwohl das Herunterfahren abgebrochen wurde, konnte es durch das Klicken auf die entsprechende Schaltfläche nicht wieder aufgenommen werden. Auch eine externe Anwendung der Systemadministratoren zum Abbrechen des Herunterfahrens brachte nicht den gewünschten Erfolg. Durch eine Internetrecherche wurde festgestellt, dass sich die Art wie das Betriebssystem beendet wird, sich bei *Windows 7* geändert hat und deshalb die Abbruchbedingung für das Herunterfahren im Programmcode anders gehandhabt werden muss. Trotz der expliziten Berücksichtigung des angewandten Betriebssystem und der intensiven Suche nach der Ursache konnte der Fehler nicht behoben werden. Daher wurde bei der Einweisung der Probranden in

[37] http://msdn.microsoft.com/en-us/library/ms591593.aspx

das Programm, erläutert, dass beim Ausschalten des PC's der Akku des Sessels aufgeladen und die *Hanappa* vom Strom getrennt werden sollte. Dies wurde ebenfalls im Benutzerhandbuch vermerkt, das jeder Proband erhalten hat, um bei Unsicherheiten, diese Punkte nachlesen zu können.

Kapitel 8

Ausblick

In diesem Kapitel werden Verbesserungen und Features erörtert, welche die Benutzeroberfläche des Prototypen betreffen. Dadurch soll eine bessere Benutzerfreundlichkeit erzielt und die Motivation der Benutzer zu einer besseren Sitzhaltung verbessert werden. Außerdem wird beschrieben, wie auf Basis der gewonnenen Erkenntnisse aus Kapitel 1, die Bürogestaltung für ein gesundes Arbeiten aussehen könnte.

8.1 Verbesserungen Prototyp

Die Benutzer erhalten durch das Rückmeldesystem im Laufe des Tages Informationen über ihr aktuelles Sitzverhalten. Für den Benutzer wäre es aber auch vorteilhaft, wenn er sein Sitzverhalten über einen längeren Zeitraum analysieren könnte. Durch das Erstellen einer Chronik des Sitzverhaltens und eine graphische Darstellung dieser, sollte im besten Fall die Fortschritte des Benutzers dargestellt werden. Im Falle einer Verschlechterung ist es dem Benutzer eventuell möglich, diese mit gewissen Ereignissen (Unfall) oder Umstände (Stress) zu verbinden und Rückschlüsse zu ziehen und somit eine Verbesserung anstreben zu können. In technischer Hinsicht muss für die Chronik ein geeignetes Dateiformat gefunden werden in dem man die Daten abspeichert. So wäre es vermutlich sinnvoll, den aktuellen Monat separat zu speichern, während man für schon vergangene Monate die Daten zusammenfasst, um so eine Auswertung der Daten zu beschleunigen. So wäre für den aktuellen Monat eine genaue Statistik verfügbar. Diese Statistik könnte eine Auswertung eines bestimmten Tages oder Zeitraumes zulassen, welche sich der Benutzer mit den Grafiken abspeichern oder ausdrucken könnte. Für die vorherigen Monate wäre nur mehr eine Auswertung der Ergebnisse die erreicht wurden im Vergleich zu den vorherigen Monaten möglich, aber keine detaillierte Statistik.

Weiters sollte das Starten der Trainingseinheit auch manuell möglich

sein, um Stress oder auftretende Verspannungen (unabhängig vom Sitzverhalten) abzubauen. Hierfür sollten auch Trainingseinheiten, welche gezielt gegen *RSI* wirken, eingeführt werden und abwechselnd zum Tragen kommen.

Eine Motivation zum besseren Sitzverhalten könnte auch durch eine Art Bonussystem erreicht werden. So könnten für langes, aktives Sitzen und eine gute absolvierte Trainingseinheit Punkte vergeben werden. Durch die Punkte könnten neue Features oder Designs freigeschaltet werden, wodurch die Software individuell angepasst werden könnte. Es wäre auch eine Verknüpfung zu sozialen Netzwerken denkbar, wo in regelmäßigen Abständen die Erfolge der Person veröffentlicht werden, wie es bei einigen auf dem Markt erhältlichen Produkten (*Nike+* [38], *runtastic* [39]) schon der Fall ist, wodurch die Motivation gesteigert werden könnte.

Neben mehr Übungseinheiten wäre es auch möglich anstatt des Videos ein Spiel zu starten. Dies erfordert auch Geschicklichkeit, da Spiele wie eine Art *Moorhuhnjagd* [26] oder für kältere Monate ein Skirennen, nur über die Bewegungen am Sessel gesteuert werden. Damit kein Konflikt mit dem Arbeitgeber entsteht, sollte dieses Spiel nur einmal am Tag gestartet werden können (vorzugsweise in der Mittagspause).

8.2 Aktives Büro

Wie schon in Kapitel 1 erläutert, wird für den Büroalltag folgende Verteilungsaktivität vorgeschlagen [10]:

- 60% Sitzen (lebendiges Sitzen),
- 30% Stehen und
- 10% Bewegung im Raum.

Somit ist es auch sinnvoll, neben der Verwendung von Sitzgelegenheiten, die ein aktives Sitzen ermöglichen, den Aspekt für einen lebendigen Büroalltag in die Gestaltung der Büroräume einfließen zu lassen. Es mag durchaus praktisch sein, wenn der Aktenschrank direkt hinter einem steht und die Unterlagen immer griffbereit sind. Doch verbannt diese Bequemlichkeit einen auf den eigenen Bürosessel. Eine Position für den Schrank, welche zum Aufstehen ermuntert, ist deshalb positiv zu sehen.

In vielen Firmen werden regelmäßige Besprechungen über den aktuellen Status abgehalten. Diese finden meist in den firmeneigenen Besprechungsräumen statt und können durchaus länger dauern, wodurch auch die Konzentration sinken kann. Für interne Abteilungsbesprechungen könnte ein

[38] http://nikerunning.nike.com/nikeos/p/nikeplus/de_DE/plus/#//dashboard/
[39] http://www.runtastic.com/

Tisch im Büro angedacht werden, welcher aber nur Stehplätze zu Verfügung stellt. Durch das Stehen könnte die Besprechungszeit reduziert werden, da das Stehen auf Dauer unbequem wird, würden die Aussagen auf das Wesentliche reduziert werden.

Eine Brainstorming-Tafel oder Whiteboard könnte nicht nur zur Produktivität einen Beitrag leisten, sondern auch zur Gesundheit der Mitarbeiter. Auch ein Stehtisch zum Verfassen von Notizen würde auch zu mehr Bewegung anregen. In Großraumbüros, könnte dieser mit einem Raumteiler separiert werden, um so für ein paar Minuten auch einen Ruheort darstellen, wo Mitarbeiter ihre Gedanken sammeln können.

Anhang A

Anhang

```
<?xml version="1.0" encoding="utf-8" ?>
<config>
  <!-- user information for statistics -->
  <user no="1" gender="f" />
  <!-- which avatar should be used: virtual, hanappa, swopper, non
  -->
  <avatar type="virtual" />
  <!--hanappa settings -->
  <comport port="3" baudrate="460800" parity="none" bit="8"
stop="one" timeout="-1" />
  <!-- training duration in seconds -->
  <training duration="30" offsetNum="15" />
  <!-- Timespan of one iteration before calculate sitting posture -->
  <measurement tInMin="3" />
  <!-- threshold for sitting posture -->
  <threshold bad="1,9" neutral="1" />
  <!-- threshold for training video 1 -->
  <fs bad="1,4" good="1" />
  ...
  <!-- input: ant; random -> for debugging without devices; -->
  <data input="random" max="3" sec="1" path="." />
  <!-- for initializing training video selection -->
  <video name="luc" />
  <!-- constants for calculating posture -->
  <constants c1="1" c2="1" c3="15" weight="0" height="0" />
</config>
```

Programm A.1: Auszug aus der XML Konfigurationsdatei

```
internal void ChangeVideoSelection()
{
   ...
   string[] words = this.selection.Split('_');
   string[] prev = this.lastSelection.Split('_');
   bool selectionChanged = false;

   if (words[0].Equals(prev[0]))
   {
      switch (words[0])
      {
         case "holger":
            this.selection = "luc_" + words[1];
         break;

         case "luc":
            this.selection = "susi_" + words[1];
         break;

         case "susi":
           this.selection = "holger_" + words[1];
         break;
       }
      selectionChanged = true;
   }
   ...
}
```

Programm A.2: Vergleich des Personennamens der aktuellen Auswahl mit der vorherigen. Bei einer Übereinstimmung wird ein anderer Name ausgewählt.

```
1  internal void ChangeVideoSelection()
2  {
3     ...
4     if (words[1].Equals(prev[1]))
5     {
6        int num = int.Parse(words[1]);
7
8        if (num < 4)
9           num++;
10       else
11          num--;
12
13       if (!selectionChanged)
14          this.selection = words[0] + "_0" + num.ToString();
15       else
16          this.selection = this.selection.Split('_')[0]
17        + "_0" num.ToString();
18    }
19    ...
20    this.lastSelection = this.GetSelection();
21
22 }
```

Programm A.3: Überprüfung der Übungsnummer der aktuellen Auswahl mit der vorherigen. Bei Übereinstimmung wird die Nummer erhöht (falls die maximale Anzahl nicht erreicht ist) oder verringert. In Zeile 14 bzw. 16 wird der Dateiname neu zusammengestellt, je nachdem, ob die zufällige Auswahl oder die adaptierte Auswahl zu tragen kommt.

Anhang B

Inhalt der CD-Rom

B.1. Diplomarbeit

Pfad: /

_DaBa.pdf .... Diplomarbeit

B.2. Abbildungen

Pfad: /images

1_EinleitungUndMotivation .... Bilder und Grafiken des Kapitels Einleitung und Motivation

3_Konzept .... Bilder und Grafiken des Kapitels Konzept

4_Studie .... Bilder und Grafiken des Kapitels Studie

5_Redesign .... Bilder und Grafiken des Kapitels Redesign

6_Iimplementierung .... Bilder und Grafiken des Kapitels Implementierung

7_Diskussion .... Bilder und Grafiken des Kapitels Diskussion

B.3. Literatur

Pfad: /references/papers

Adamczyk – 2004.pdf .... *If not now, when? The effects of interruption at different moments within task execution*

Altmann – 2004.pdf.................. *Task interruption: Resumption lag and the role of cues*

Andrews – 2009.pdf.................. *Recovering from interruptions: Does alter type matter?*

Arroyo – 2002.pdf .... *Interruptions as multimodal outputs: Which are the less disruptive?*

Arroyo – 2003.pdf .... *Self-Adaptive multimodal-interruption interfaces*

Bailey – 2000.pdf...................... *Adjusting windows: Balancing information awareness with intrusion*

Bailey – 2001.pdf...................... *The effects of interruptions on task performance, annoyance, and anxiety in the user interface*

Beach – 2005.pdf...................... *Effects of prolonged sitting on the passive flexion stiffness of the in vivo lumbar spine*

Berendonk – 2002.pdf.............. *Maus verursacht RSI-Syndrom*

Burmistrov – 1997.pdf .... *Interruptions in the computer aided office work: Implications to user interface design*

Brumistrov – 2003.pdf .... *Do interrupted users work faster or slower? The micro-analysis of computerized text editing task*

Cades – 2006.pdf...................... *Mitigating disruptions: Can resuming an interrupted task be trained?*

Czerwinski – 2000_1.pdf.......... *Instant messaging: Effects of relevance and timing*

Czerwinski – 2000_2.pdf.......... *Instant messaging and interruptions: Influence of task type on performance*

Daian – 2007.pdf...................... *Sensitive chair: A force sensing chair with multimodal real-time feedback via agent*

Haller – 2011.pdf...................... *Finding the right way for interrupting people improving their sitting posture*

Jafarinaimi – 2005.pdf.............. *Breakaway: An ambient display designed to change human behavior*

Kingma – 2009.pdf .... *Static and dynamic postural loadings during computer work in females: Sitting on an office chair versus sitting on an exercise ball*

Kuhnt – 2003.pdf .... *Präventive Rückenschule – Teilnehmerunterlagen*

Ludwig – 2008.pdf .... *Untersuchung zur Änderung der Oberkörperdurchblutung während des Sitzens auf Stühlen mit beweglicher Sitzfläche*

Mark – 2008.pdf .... *The cost of interrupted work: More speed and stress*

Petersen – 2006.pdf .... *Bildschirmarbeitsplätze – Eine arbeitsmedizinische Bewertung*

Schrempf – 2011.pdf .... *PostureCare – Towards a novel system for posture monitoring and guidance*

B.4. Onlinequellen

Prad: /references/online/literature

*.pdf .... Kopie der im Literaturverzeichnis angegebenen Onlinequellen

Prad: /references/online/others

*.pdf .... Kopien verschiedenster Webseiten, die in dieser Arbeit verlinkt wurden

Literaturverzeichnis

[1] Adamczyk, P. D. und Bailey, B. P.: *If not now, when?: the effects of interruption at different moments within task execution.* In: *Proceedings of the SIGCHI conference on Human factors in computing systems.* CHI '04. Vienna, Austria, ACM, 2004, S. 271–278. URL: http://doi.acm.org/10.1145/985692.985727.

[2] Altmann, E. M. und Trafton, J. G.: *Task interruption: Resumption lag and the role of cues.* In: *Proceedings of the 26th Annual Conference of the Cognitive Science Society*, Chicago, USA, Associates, 2004, S. 43–48.

[3] Andrews, A. E. et al.: *Recovering from interruptions: Does alter type matter?* In: *Proceedings of the Human Factors and Ergonomics Society 53rd Annual Meeting (HFES'09),* San Antonio, TX, USA, 2009, S 409–413.

[4] Arroyo, E. et al.: *Interruptions as Multimodal Outputs: Which are the Less Disruptive?* In: *Proceedings of the 4th IEEE International Conference on Multimodal Interfaces*, Pittsburgh, PA, USA, ACM, 2002, S. 479–482.

[5] Arroyo, E. und Selker, T.: *Self-adaptive multimodal-interruption interfaces.* In: *Proceedings of the 8th international conference on Intelligent user interfaces.* IUI '03, ACM, 2003, S. 6–11. URL: http://doi.acm.org/10.1145/604045.604051.

[6] Bailey, B. P. et al.: *Adjusting windows: Balancing information awareness with intrusion.* In: *Proceedings of the 6th Conference on Human Factors and the Web.* Austin, USA, 2000. URL: https://agora.cs.uiuc.edu/download/attachments/9644168/hfweb-2000.pdf?version=1.

[7] Bailey, B. P. et al.: *The effects of interruptions on task performance, annoyance, and anxiety in the user interface.* In: *Proceedings of INTERACT* 01, IOS Press, 2001, S. 593–601.

[8] Beach, T. A. C. et al.: *Effects of prolonged sitting on the passive flesion stiffness of the in vivo lumbar spine. The Spine Journal of the North American Spine Society volume 5(2)*. Elsevier, 2005, S. 145–154. URL: http://www.ncbi.nlm.nih.gov/pubmed/15749614.

[9] Berendonk, U.: *Maus verursacht RSI-Syndrom*. Bundesverwaltungs-amt D, Info 1691 2002, http://www.dji.de/itgruppe/bva.pdf.

[10] Breithecker, D.: Haltung und Bewegung. [Online]. http://www.haltungbewegung.de/körperliche-und-geistige-gesundheit-braucht-einen-haltungswechsel.aspx.

[11] Burmistrov, I. und Leonova, A.: *Do interrupted users work faster or slower? The micro-analysis of computerized text editing task*. In: *Human-Computer Interaction: Theory and Practice (Part I) - Proceedings of HCI International 2003,* Lawrence Erlbaum Associate,, 2003 S. 621–625. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.7003&rep=rep1&type=pdf.

[12] Burmistrov, I. und Leonova, A.: *Interruptions in the computer aided office work: Implications to user interface design*. In: *From Experience to Innovation: Proceedings of the 13th Triennial Congress of the International Ergonomics Association, Vol. 5*. Helsinki, FIN: Finnish Institute of Occupational Health, 1997, S. 77–79.

[13] Cades, D. M. et al.: *Mitigating disruptions: Can resuming an interrupted task be trained?* In: *Proceedings of the Human Factors and Ergonomics Society 50th Annual Meeting*, Human Factors and Ergonomics Society, 2006, S. 368–371. URL: http://www.ingentaconnect.com/content/hfes/hfproc/2006/00000050/00000003/art00033.

[14] Czerwinski, M. et al.: *Instant Messaging and Interruption: Influence of Task Type on Performance*. In: *OZCHI 2000 Conference Proceedings*, Association for Computing Machinery, 2000, S. 356–361.

[15] Czerwinski, M. et al.: *Instant Messaging: Effects of Relevance and Timing*. In: *People and Computers XIV: Proceedings of HCI 2000*, 2000, S. 71–76.

[16] Daian, I. et al.: *Sensitive chair: a force sensing chair with multimodal real-time feedback via agent*. In: *Proceedings of the 14th European conference on Cognitive ergonomics: invent! explore! London, UK,* ACM, 2007, S. 163–166. URL: http://doi.acm.org/10.1145/1362550.1362583.

[17] Egoscue, P. und Gittines, R. : *Pain free at your PC.* Bantam Books, 1999, ISBN: 0-553-38052-4.

[18] Ertel, M. et al.: *Auswirkungen der Bildschirmarbeit auf Gesundheit und Wohlbefinden (Schlussbericht)*. D. Wirtschaftsverlag NW, 2000, ISBN: 3-89429-864-2.

[19] Haller, M. et al.: *Finding the right way for interrupting people improving their sitting posture*. In: *INTERACT 2011, 13th IFIP TC13 Conference on Human-Computer Interaction*: Lisbon, P, Springer, 2011, S. 1–17, http://mi-lab.org/files/2011/03/sitz-gscheit-final.pdf.

[20] Jafarinaimi, N. et al.: *Breakaway: an ambient display designed to change human behavior*. In: *CHI '05 extended abstracts on Human factors in computing systems*. Portland, OR, USA, ACM, 2005, S. 1945–1948. URL: http://doi.acm.org/10.1145/1056808.1057063.

[21] Kingma, I. und Van Dieën, J. H.: *Static and dynamic postural loadings during computer work in females: Sitting on an office chair versus sitting on an exercise ball. pplied Ergonomics volume 40(2)*, 2009, S. 199–205. URL: http://www.ncbi.nlm.nih.gov/pubmed/15749614.

[22] Kuhnt, U.: *Präventive Rückenschule – Teilnehmerunterlagen*. 2003.

[23] Ludwig, O. und Breithecker, D.: *Untersuchung zur Änderung der Oberkörperdurchblutung während des Sitzens auf Stühlen mit beweglicher Sitzfläche. Haltung und Bewegung*, Heft 3, 2008, S. 5–12.

[24] Mark, G. et al.: *The cost of interrupted work: more speed and stress.* In: *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, Florence, Italy. ACM, 2008, S. 107–110.

[25] Petersen, J.: *Bildschirmarbeitsplätze –eine arbeitsmedizinische Bewertung. Deutsches Ärzteblatt*, Heft 30 (Juli), 2006. URL: http://www.aerzteblatt.de/v4/archiv/artikel.asp?id=54036.

[26] Schrempf, A. et al.: *PostureCare –towards a Novel System for Posture Monitoring and Guidance*. Erscheint in *18th World Congress of the International Federation of Automatic Control (IFAC)*, 2011, http://mi-lab.org/files/2011/04/postureCare.pdf.